Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Využitie všeobecného počítačového hráča v spoločenských hrách

Bakalárska práca

2021
Vladimír Bačinský

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Využitie všeobecného počítačového hráča v spoločenských hrách

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	RNDr. Jozef Šiška PhD.

Bratislava, 2021
Vladimír Bačinský

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov:

Anotácia:

Vedúci:
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 11.12.2017

Dátum schválenia: 12.12.2017 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Tu môžete poďakovať školiteľovi, prípadne ďalším osobám, ktoré vám s prácou nejako pomohli, poradili, poskytli dáta a podobne.

Abstrakt

Kľúčové slová:

Abstract

Keywords:

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Napísať na konci.

> Úvod je prvou komplexnou informáciou o práci, jej cieli, obsahu a štruktúre. Úvod sa vzťahuje na spracovanú tému konkrétne, obsahuje stručný a výstižný opis problematiky, charakterizuje stav poznania alebo praxe v oblasti, ktorá je predmetom školského diela a oboznamuje s významom, cieľmi a zámermi školského diela. Autor v úvode zdôrazňuje, prečo je práca dôležitá a prečo sa rozhodol spracovať danú tému. Úvod ako názov kapitoly sa nečísluje a jeho rozsah je spravidla 1 až 2 strany.

Kapitola 1

Úvod do problematiky

V tejto kapitole si zadefinujeme základné pojmy, ktoré budeme používať v tejto bakalárskej práci. Predstavíme si, čo je to Všeobecný Počítačový Hráč a oboznámime sa s najdôležitejším nástrojom práce a to Jazykom Popisu Hry.

1.1 Základné pojmy

Neohodnotený graf je dvojica G = (V, E), pričom V je množina elementov, ktoré nazývame vrcholy a E je množina dvojíc vrcholov, ktoré nazývame hrany. [1]
Strom je súvislý acyklický graf, teda neobsahuje kružnicu a medzi každou dvojicou vrcholov existuje cesta.

1.2 Všeobecný Počítačový Hráč

Pre mnoho spoločenských hier existujú programy a algortimy, ktoré nám umožňujú ich hranie. Napríklad ľahko nájdeme program hrajúci šach či dámu. Ak by sme chceli program, ktorý umožňuje hranie aj dámy aj šachu, tak tento program by potreboval vedieť pravidlá oboch hier dopredu. Práve najväčšou výhodou Všeobecného Počítačového Hráča (označujeme GGP, z anglického General Game Playing) je, že dokáže hrať hry napísané v Jazyku Popisu Hry (označujeme GDL z anglického Game Description Language) bez toho, aby poznal ich pravidlá vopred. GGP je teda schopný hrať viacero druhov hier, vrátane dvoch vyššie spomenutých. Musia však mať spoločnú štruktúru a to konkrétne hra musí mať konečný počet hráčov, konečný počet stavov a konečný počet akcií v ktoromkoľvek stave. Navyše jeden zo stavov musí byť počiatočný [3]. Aby hra bola hrateľná, tak musí existovať aspoň jeden taký stav, ktorý je koncový a zároveň existuje taká postupnosť legálnych ťahov, že pomocou nich sa vieme dostať z počiatočného stavu do koncového. Formálne môžeme reprezentovať model hry nasledovne:
Nech n je počet hráčov a S je množina stavov, pričom

- $A_1, \ldots, A_n$ - A_i je konečná množina akcií (legálnych ťahov) i-teho hráča, i patrí $\{1, \ldots, n\}$
- $I_1, \ldots, I_n$ - I_i je podmnožina $A_i \times S$ - teda dvojice akcia i-teho hráča a stav, i patrí $\{1, \ldots, n\}$
- $f : S \times A_1 \times \ldots \times A_n \longrightarrow S$ - funkcia na aktualizovanie, teda slúži na prechod z jedného stavu do druhého, pomocou legálnych ťahov hráčov
- s_1 patrí S - s_1 je počiatočný stav
- $T \subseteq S$ - množina terminálnych, teda koncových stavov

Takýto formálnejší model hry využijeme neskôr pri demonštrácií konkrétných algoritmov, ktoré GGP hráči využívajú. Všeobecní počítačoví hráči sa líšia stratégiou, pomocou ktorej vyberajú akcie, teda legálne ťahy v aktuálnom stave. K najtriviálnejším patria Legálny Hráč (značíme LegalPlayer) a Náhodný Hráč (značíme RandomPlayer). LegalPlayer funguje jednoducho a to tak, že z pomedzi zoznamu legálnych ťahov vyberie prvý v poradí. RandomPlayer hrá taktiež elementárne a zo zoznamu akcií vyberá náhodne. Existujú však aj omnoho komplikovanejší hráči, ktorí pri vyberaní ťahu využívajú efektívne algoritmy, ktoré im pomáhajú vybrať najoptimálnejšiu akciu a práve s niektorými z nich sa oboznámime. V našej práci sa budeme zaoberať hrou, ktorá je určená pre dvoch hráčov a je konkurenčná, teda víťazom sa stáva len jeden hráč. Narozdiel od hier určených pre jedného hráča, v tých viacnásobných výber najlepšieho ťahu je závislý od potenciálneho budúceho ťahu protihráča. Navyše žiaden hráč nevie priamo odhadnúť ťah, ktorý súper zahrá, preto musíme počítať so všetkými možnými akciami protihráča. Celú hru potom môžeme reprezentovať ako strom, pričom koreňom je počiatočný stav, vnútorné vrcholy sú stavy, ku ktorým sa vieme pomocou funkcie f dostať z počiatočného stavu a listy sú terminálne stavy a orientované hrany reprezentujú prechod medzi stavmi pomocou funkcie f. Jednoduchá reprezentácia by mohla vyzerať ako na obrázku 1.1. Počet koncových stavov sa rovná počtu všetkých možných konfigurácií danej hry a ich hodnota sa rovná počtu dosiahnutých bodov v tomto stave. Na výber optimálneho ťahu v aktuálnom stave existuje napríklad algoritmus nazývaný Minimax.

Minimax

Jeho funkcionalitu budeme demonštrovať na príklade, kde hráč zahrá jeden ťah a po ňom nasleduje hneď súper, teda ak je hráč na ťahu v hĺbke h, tak v hĺbke h+1 bude na ťahu súper. Na pochopenie jeho funkčnosti nám pomôže obrázok 1.2. Aby Minimax fungoval, potrebuje mať ohodnotené listy v strome, teda v našom

prípade to sú ohodnotené terminálne stavy. Následne sa hráč na ťahu v aktuálnom stave snaží pomocou Minimaxu maximalizovať svoj optimálny legálny ťah, za predpokladu, že súper si vždy vyberie pre seba taktiež najlepší ťah. Algoritmus minimax začína počítať v o jedna menšej hĺbke celého stromu. Všetky stavy v tejto hĺbke reprezentujú ťah súpera. Pre protihráča je teda najefektívnejšie vybrať minimum z hodnôt svojich synov (vrcholy, do ktorých vedie hrana z daného vrcholu) v o jedna väčšej hĺbke a naopak pre nás je najlepšie vybrať maximum zo synov. Analogicky postupným znižovaním hĺbky a striedaním maxima a minima v každej hĺbke sa dostaneme až k aktuálnemu stavu, kde sme dostali najväčšie číslo z vrcholov v o jedna väčšej hĺbke, teda najoptimálnejší ťah je práve do vrchola s najväčšou hodnotou, ktorú minimax vypočítal.

Nevýhodou tohto algoritmu je, že zakaždým musí prehľadať celý strom, preto existuje vylepšený Minimax.

Optimalizovaný Minimax

V prípadoch takých stromov, kde poznáme absolútnu maximálnu respektíve minimálnu možnú hodnotu v terminálnych stavoch, tak nebudeme musieť vždy prehľadávať celý strom. Napríklad ak vieme, že absolútna minimálna hodnota je 0 a zároveň algoritmus Minimax v stave, kedy počíta minimum svojich synov (označujeme takýto stav minnode) má v jednom z nich 0, tak logicky už nemusí hľadať v ďalších synoch, pretože menšiu hodnotu už nenájde, teda vráti 0. Analogicky ak vieme, že napríklad absolútna maximálna hodnota v koncových stavoch je 100, tak ak Minimax v stave, kde počíta maximum (označujeme maxnode) nájde v jednom zo svojich synov hodnotu 100, rovnako už nemusí hľadať ďalej, pretože väčšiu hodnotu už nenájde a preto vráti 100. Tento optimalizovaný Minimax nám vie pomôcť v niektorých prípadoch, avšak ukážeme si ešte silnejšiu verziu tohto optimalizovaného minimaxu a to Alfa-Beta vyhľadávanie.

Alfa-Beta vyhľadávanie

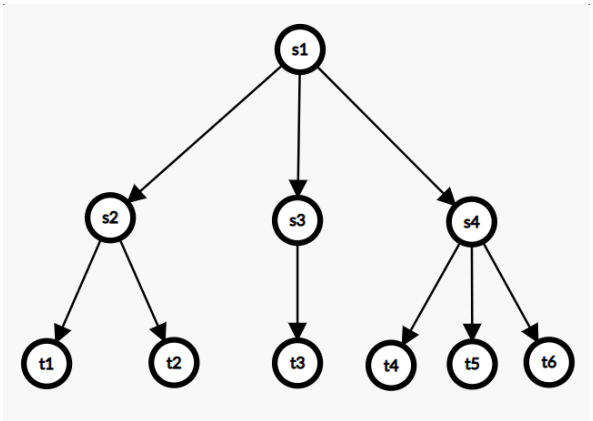
Tento algoritmus si počas behu udržiava doposiaľ najmenšiu nájdenú hodnotu nazývanú alfa (označujeme α) a najväčšiu nájdenú hodnotu nazývanú beta (označujeme β). Algoritmus môžeme rozdeliť na dva prípady, jeden ak vylúčime vetvu stromu v minnode a druhý v maxnode. Rozoberme si teda prvú možnosť a to ak v minnode nájde Minimax v synovi menšiu hodnotu ako alfa (označme túto hodnotu x), tak potom už nebude musieť prehľadávať ďalších synov a jednoducho vráti hodnotu α. Ak by aj v ďalších synoch bola väčšia hodnota ako x, tak tá je pre nás irelevantná, pretože minnode aj tak vyberie minimum, teda α. Čo v prípade, ak v ďalších synoch je menšia hodnota ako x? Tá sa taktiež javí ako zbytočná, pretože vieme, že neskôr v o jedna menšej hĺbke bude následne maxnode vyberať maximálnu hodnotu zo synov a tam vždy vyberie α pred tou

našou menšou hodnotou v synovi. Tým sme ukázali, že obidva prípady synov, už nebudeme potrebovat a preto túto vetvu nebudeme musieť ďalej prehľadávať. Analogicky v druhej možnosti, ak v maxnode nájde v synovi hodnotu väčšiu ako beta (označme túto hodnotu y). V prípade, že v ďalších synoch je menšie číslo ako y, tak je táto vetva zbytočná, pretože maxnode vyberie maximum. Ak je v neprehľadaných synoch väčšia hodnota ako y, tak tá je taktiež nepodstatná, pretože v ďalšom kroku bude Minimax v o jedna menšej hĺbke, teda bude v minode vyberať minimum a tam opäť zvíťazí stav s hodnotou y. Neskôr v tejto bakalárskej práci budeme testovať spoločenskú hru práve na GGP hráčov a jedným z nich je hráč, ktorý využíva Monte Carlo Vyhľadávanie.

Monte Carlo Vyhľadávanie

Tento algoritmus označovaný tiež ako MCS (z anglického Monte Carlo Search) sa snaží nájsť rovnako ako vyššie spomínané algoritmy optimálny legálny ťah pre hráča na ťahu v aktuálnom stave. Pre lepšie pochopenie nám pomôže obrázok 1.3. Ak Všeobecný Počítačový Hráč hrá hru napísanú v GDL, tak má vždy stanovený nejaký časový limit, kým sa musí rozhodnúť pre výber svojho ťahu. MCS spustí simuláciu postupne pre každú možnú akciu v danom stave, teda v každom synovi. V simulácii algoritmus vyberá ťahy náhodne až kým sa nedostane k terminálnemu stavu s hodnotou totožnou s počtom dosiahnutých bodov na konci hry. Túto hodnotu si uloží ako ohodnotenie daného syna a pokračuje z aktuálneho stavu analogicky ďalšími synmi. Ak prejde všetkých synov, tak simuláciu znovu opakuje chronologicky od najľavejšieho syna a postup opakuje s takým rozšírením, že hodnotu ktorú našiel v koncovom stave pripočíta k danej hodnote syna a zároveň si zapamätá počet simulácii z daného stavu. Ak sme dosiahli časová limit, tak postupne MCS v každom synovi predelí jeho hodnotu počtom opakovaní a následne vyberie z týchto hodnôt maximum. Hrana, ktorá smeruje k tomuto maximu je náš optimálny ťah, ktorý MCS vyberie.

Ukázali sme si základné algoritmy, ktoré všeobecní počítačoví hráči môžu využívať pri vyberaní najoptimálnejšieho ťahu v aktualnom stave. Avšak programátori sa vždy snažia stratégie týchto všeobecných hráčov zlepšovať a dostávame sa k zaujímavosti v oblasti GGP, kde títo programátori a všeobecní počítačoví hráči môžu navzájom súťažiť. Už od roku 2005 sa na konferencii AAAI (z anglického Association for the Advancement of Artificial Intelligence) konajú každoročné GGP súťaže. V prvej časti súťaže sa zúčastnení hodnotia na základe ich schopnosti vykonávať legálne ťahy, získavať prevahu a čo najefektívnejšie dokončiť hry. V druhom kole sa účastníci stretávajú proti sebe v čoraz zložitejších hrách. Program, ktorý v tejto fáze vyhráva najviac hier, tak víťazí v súťaži a do roku 2013 jej tvorca získaval dokonca cenu 10 000 dolárov [2].



Obr. 1.1: Reprezentácia hry stromom, pričom vrchol v hĺbke 0 je počiatočný stav, vrcholy v hĺbke 1 sú stavy, ktoré vznikli prechodom z počiatočného stavu pomocou funkcie f a vrcholy v hĺbke 1 sú koncové stavy. Šípky v tomto strome znázorňujú jednotlivé prechody medzi stavmi a to na základe funkcie f, teda legálnych ťahov každého hráča v danom stave.

Neskôr v tejto práci vyskúšame rôzne implementácie práve GGP hráčov na vybranej spoločenskej hre.

1.3 Jazyk Popisu Hry

Jazyk popisu hry bol vyvinutý, aby vykonávanie logických úloh pre všeobecné hranie hier bolo efektívne. GDL popisuje stavy hry ako množinu faktov a množinu logických pravidiel [4]. Napríklad ak máme fakty

$$rodic(jozef, fero)$$
$$rodic(fero, julka)$$

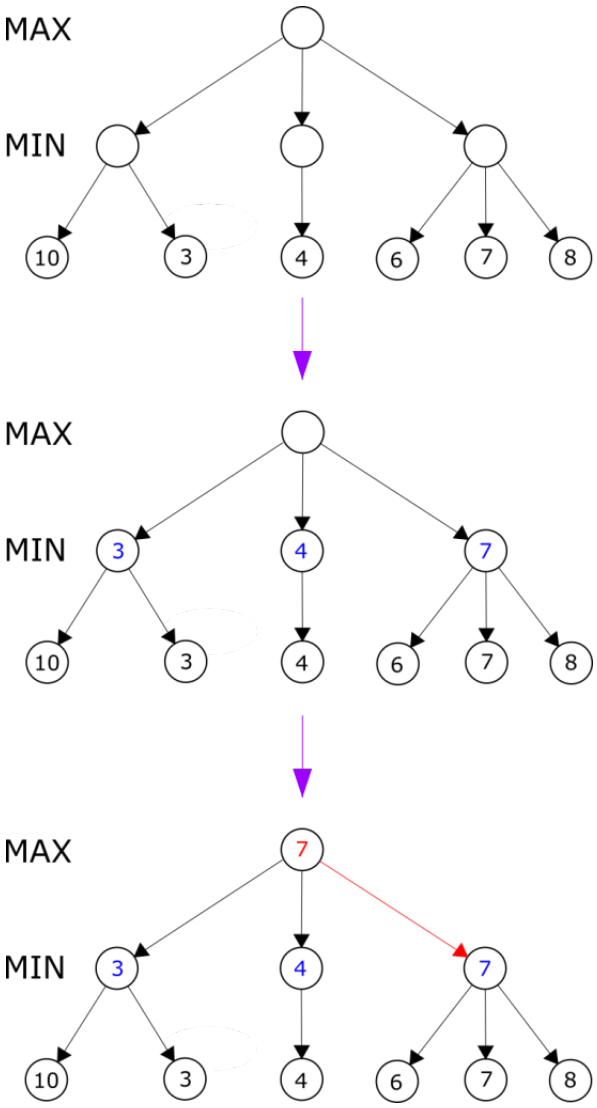
tak je faktom, že Jozef je rodičom Fera a Fero je rodičom Julky. Ak zároveň máme pravidlo

$$staryrodic(X, Z) :- \quad rodic(X, Y), rodic(Y, Z)$$

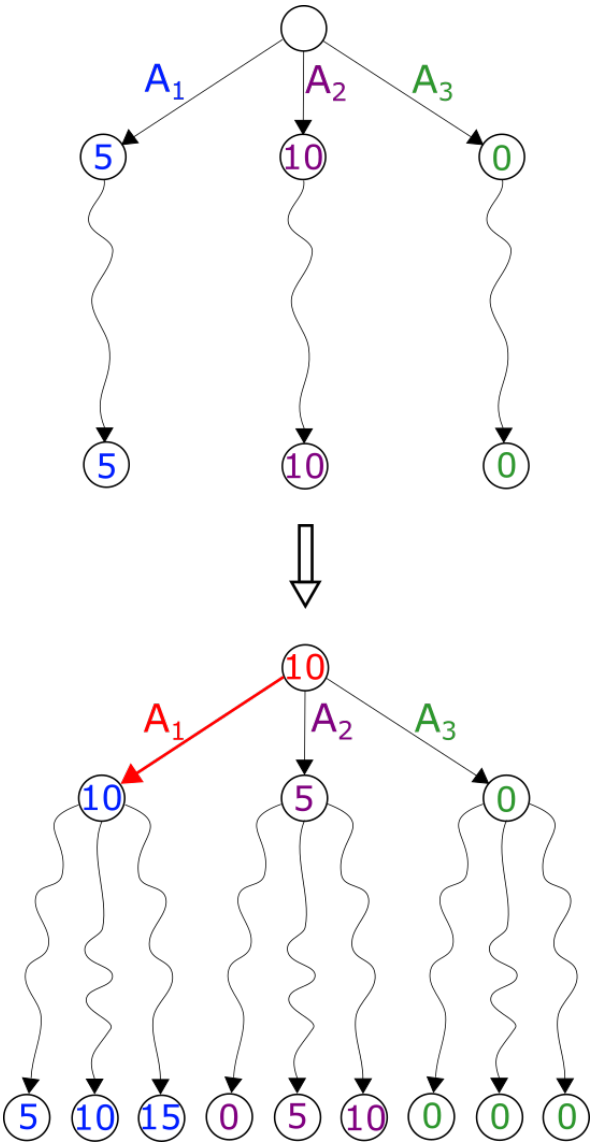
ktoré hovorí, že ak X je rodičom Y a zároveň Y je rodičom Z, tak potom X je starým rodičom Z, tak z našich faktov a pravidla môžeme odvodiť

$$staryrodic(jozef, julka)$$

teda Jozef je starý rodič Julky.
Zadefinujme si teraz formálnejšie základné pojmy používané v GDL, term a pravidlo v GDL, ktoré sú podobné ako v [4]. Medzi slovnú zásobu v GDL patria objektové konštanty (označujeme písmenami zo začiatku abecedy: a, b, c), funkčné konštanty (označujeme malými písmenkami abecedy: f, g, h), relačné konštanty (označujeme malými písmenkami abecedy: p, q, r) a premenné (označujeme veľkými písmenami, prevažne z konca abecedy: X, Y, Z).



Obr. 1.2: Pomocou Minimaxu chceme zistiť aký je pre nás najlepší ťah v aktuálnom stave, teda na obrázku v koreni. Listy v tomto strome sú terminálne stavy ohodnotené konkretným počtom bodov, ktoré vieme získať. Stavy v hĺbke 1, teda vnutorné stavy repreznetujú stavy, kedy je na ťahu súper. Minimax začína svoj výpočet v hĺbke o jedna menšej ako sú listy, v našom prípade tena v hĺbke 1. V každom vrchole tejto hĺbky vypočíta minimálnu hodnotu svojich synov a vloží ju ako hodnotu vrchola. Minimum, pretože súper sa snaží minimalizovať počet bodov, ktoré môžeme získať. Následne sa algoritmus minimax posunie o hĺbku nižšie a vypočíta v každom vrchole tejto hĺbky, teda v našom prípade len v koreni maximum svojich synov, pretože my chceme maximalizovať náš bodový úspech. Tento postup minimax analogicky opakuje až kým sa nedostane ku koreňu stromu, čo sme v našom prípade už dosiahli. Najoptimálnejší ťah pre nás je teda legálny ťah reprezentovaný červenou (doprava) šípkou.



Obr. 1.3: V hornej časti obrázka môžeme vidieť orientovaný graf, pričom koreňom je aktuálny stav, šípky z koreňa do vnútorných vrcholov znázorňujú prechod z aktuálneho stavu zahraním jedného z legálnych ťahov $A_1, A_2 a A_3$. Vnútorne vrcholy reprezentujú následny ťah súpera a šípky do listov, teda terminálnych stavov, znázorňuju nejakú možnú konfiguráciu. V tomto prípade je to postupnosť náhodných ťahov. Tento obrázok konkrétne demonštruje fázu, kedy MCS simuloval náhodne ťahy cez všetkých troch synov jedenkrát. Hodnotu, ktorú našiel v koncovcých stavov si uložil do synov. Následne na druhom obrázku je ukážka pridaných ďalších dvoch simulácii pre každého syna, ale hodnota synov je iná a vypočítala sa nasledovne. Súčet troch listov do ktorých sme sa dopracovali z daného syna nejakou konfiguráciou predelíme počtom simulácii. MCS potom už len vyberie maximum z týchto synov a práve hrana, ktorá smeruje do maxima je náš optimálny ťah.

Termom v GDL sú objektové konštanty, premenné alebo funkčné konštanty pozostavajúce z konečného počtu konštánt a premenných (označujeme $f(a, X, b)$). *Atóm* je relácia pozostávajúca z konštánt a termov - označujeme $p(a, f(a, b))$. Negácia je výraz, ktorý sa skladá zo znakov negácie a atómu - označujeme $\neg p(a, f(a, b))$. Literál je atóm alebo negácia atómu - označujeme $p(a, f(a, b))$ alebo $\neg p(a, f(a, b))$. Inštancia v hlave je pravdivá vtedy, ak všetky pozitívne literály sú pravdivé (true) a všetky negatívne literály sú nepravdivé (false).

Pravidlo v GDL sa skladá z hlavy a zoznamu literálov, ktoré sú oddelené čiarkou, označujeme

$$h :- \quad l_1, ..., l_n$$

Pravidlo vyhodnocujeme ako konjunkciu [1] literálov. Zoznam literálov nazývame telo. Ešte pred tým ako si prejdeme postupne všetky základné relácie Jazyka Popisu Hry, tak si ukážeme konkrétnu syntax v GDL nazývanú formát výmeny vedomostí (označujeme KIF z anglického Knowledge Interchange Format). V KIF pred premenné píšeme otáznik, teda premmenna X je písaná ako `?x`. Funkčnú konštantu $f(a, X)$ píšeme ako `(f a ?x)` alebo $true(control(X))$ píšeme ako `(true (control x))`. Negáciu $\neg(true(control(X)))$ píšeme ako `(not (true (control x)))` a pravidlo $p(X) \Leftarrow q(X) \wedge \neg r(X)$ píšeme ako `(<= (p ?x) (q ?x) (not (r ?x)))` [4]. Začiatok komentárov značíme bodkočiarkou.

Medzi základné relácie v GDL patria: *role, true, init, next, legal, does, goal a terminal.* Poďme si ich postupne všetky vysvetliť. Príklady budeme demonštrovať na jednoduchej hre piškvorky 3x3 (Tic-Tac-Toe). Napriek tomu, že takmer všetci túto hru poznáme, tak pre istotu si pripomeňme jej pravidlá. Hru hrajú dvaja hráči (označme hráč x a hráč o) na hracej ploche štvorca, v ktorom je 9 menších štvorcov, teda 3x3. Ťahy hráčov spočívajú v položení, respektíve nakreslení svojho znaku na konkrétny štvorec, pričom po nakreslení nasleduje ťah druhého hráča. Na každom políčku hracej plochy môže byť maximálne jeden znak. Cieľom hry je získať trojicu svojich znakov, buď v niektorom z riadkov, stĺpcov alebo na diagonálach. V prípade, že niektorý z hráčov získa spomínanú trojicu, tak vyhráva a hra končí.

Predikát *role* definuje hráčov hry, teda `(role x)` znamená, že jedným z hráčov tejto hry je x.

Predikát *true* (*true(fact)*) popisuje skutočnosti, ktoré sú pravdivé v aktuálnom stave. Potom `(true(cell 1 1 b))` znamená, že na hracej ploche na pozíciii 1 1 v bunke je blank [2].

Predikát *init* je analogický s *true*, ale je určený len pre začiatok hry, teda pre počiatočný stav. Príkladom v piškvorkách tohto predikátu je `(init(cell x y b))` $\forall x, y \in$

[1] konjunkcia (označujeme $\wedge$) je operátor logickej spojky. Množina operandov *and* je pravdivá, ak všetky operandy sú pravdivé

[2] prázdnota, prázdne miesto

$\{1, 2, 3\}$, pretože na začiatku Tic-Tac-Toe sú všetky bunky hracej plochy prázdne.
Relácia *next* je analogická s *true*, ale predstavuje fakty, ktoré budú platiť v nasledujúcom stave. Nech `(control x)` znamená, že na ťahu je hráč x, potom `(<= (next (control o)) (true (control x)))` znamená, že v nasledujúcom stave bude platiť, že hráč o bude na ťahu, ak je pravda, že hráč x je na ťahu v aktuálnom stave.
Existuje špeciálna akcia *noop*, ktorá indikuje, že hráč neurobil žiadnu akciu.
Relácia *legal*: $(legal(hrac, krok))$ predstavuje legálne kroky hráčov v aktuálnom stave. Napríklad `(<= (legal x noop) (true (control o)))`, čiže legálnym krokom pre hráča x je akcia *noop*, ak je práve na ťahu druhý hráč o.
Relácia *does* indikuje kroky vykonané hráčmi na ťahu. Nech `(mark x y)` je faktom, ktorý znamená krok zaznačenia (v reálnej papierovej verzii nakreslil svoj znak) na pozícii x y, potom pravidlo `(<= (next (cell 1 1 x)) (does x (mark 1 1)))` hovorí o tom, že v ďalšom stave hry bude na pozícii 1 1 v bunke symbol `x`, ak platí mark, teda že hráč x zaznačil na pozícii 1 1.
Predikát *goal* určuje odmenu pre hráčov, napríklad za dokončenie úlohy alebo ako v Tic-Tac-Toe za výhru. Nech `(line ?player)` je true práve vtedy ak je nejaký riadok, stĺpec alebo diagonály zaplnený rovnakými znakmi alebo po diagonálach, potom `(<= (goal ?player 100) (line ?player))` znamená, že pre hráča, pre ktorého platí *line*, tak dostáva 100 bodov.
Relácia *Terminal* definuje koniec hry. Nech *open* hovorí o tom či je ešte nejaký blank na hracej ploche, potom `(<= terminal (not open))` hovorí, že koniec hry nastane napríkald vtedy, ak už na hracej ploche nie je bunka s blankom.

Kapitola 2

Spoločenská hra "Slovania obchodníkmi"

V tejto kapitole podrobne rozoberieme spoločenskú hru s názvom Slovania obchodníkmi, ktorú som sám vytvoril. Popíšeme si akého typu táto hra je, z akých pravidiel sa skladá, čo je jej cieľom, aké má špecifické prvky a podobne.

2.1 Úvod k hre

Slovania obchodníkmi je typická strategická stolová hra. Hrá sa na konkrétnom hracom pláne, po ktorom sa hýbu figúrky hráčov podľa nejakých pravidiel. Hrací plán je špecifický pre tento konkrétny druh hry a vždy je rovnaký. Je podobný hre "Človeče nehnevaj sa", ale s tým rozdielom, že figúrky hráčov sa môžu hýbať oboma smermi. V princípe hracím plánom je jednoduchý neohodnotený graf. Ak hráč hodil na kocke číslo 6, tak si následne môže vybrať každý vrchol vo vzdialenosti práve 6 alebo na hrad/mesto vo vzdialenosti menej ako 6. Hráč nemá úplnú kontrolu nad pohybom svojej figúrky, pretože hádže kockou a tým tu vstupuje činiteľ náhody, teda táto hra patrí k nedeterministickým. Úlohou každého hráča je plniť misie. Misia sa skladá zo začiatočného mesta alebo hradu, kde konkrétny tovar musí hráč vyzdvihnúť a končiaceho mesta alebo hradu, kde zásielku doručí. Každá misia je obodovaná podľa dĺžky najkratšej cesty (teda počtu políčok najkratšej ceste). Napríklad môžeme mať misiu, ktorej úlohou je doručiť kožu z Šarišského hradu do hradu Šašov, pričom počet políčok najkratšej cesty je 20 a odmenou je 5 korún. Plnenie misií však v tejto hre nie je také jednoduché, pretože hráči si môžu kúpiť negatívne (škodiace) žetóny, ktoré neskôr môžu vkladať na políčka ciest.
Cieľom každého hráča je získať 20 bodov pomocou plnenia misií.

2.2 Pravidlá

Hra je určená pre 2 hráčov. Na ťahu je vždy práve jeden hráč. Jeho ťah sa začína hodením kocky (označme hodnotu čísla hodeného na kocke x), následne si hráč musí vybrať políčko kam presunie svoju figúrku, avšak môže ju posunúť len na políčka cesty, ktoré sú vo vzdialenosti presne x od aktualnej pozície figúrky a políčka hradov, ktoré sú vo vzdialenosti maximálne x. Potom hráč môže nakupovať negatívne žetóny, ktoré nemusí nutne v tom istom ťahu klásť na hraciu plochu alebo pozitívne žetóny. Slovania obchodníkmi sa skladajú z týchto prvkov:

Hracia plocha

Je neohodnotený graf, v ktorom vrcholy sú všetky hrady, mestá a políčka ciest a hrany sú medzi tými vrcholmi, ktoré sú susedné. Ak napríklad hrad A má vstupnú aj výstupnú jedinú cestu, tak hrana je medzi vrcholom (hradom) A a susedným políčkom cesty.

Misie

Každá misia sa skladá zo začiatočného (A) a konečného (B) hradu/mesta, počtu políčok najkratšej cesty z A do B, odmeny a názvu predmetu, konktrény príklad v tabuľke 2.1.

Negatívne žetóny

Negatívne žetóny sa nemôžu vkladať na políčka hradov alebo miest. Sú určené len na políčka ciest. Na každom políčku cesty môže byť maximálne jeden negatívny žetón. Žetóny sú 4 farieb, pretože chceme aby boli medzi hráčmi odlíšiteľné. Ak hráč stúpi na políčko, na ktorom je cudzí negatívny žetón, tak sa negatívny žetón zoberie z tohto políčka preč. Negatívne žetóny sú 3 typov:

- Z-Zbojníci: Ak hráč vstúpi na políčko, na ktorom je cudzí žetón Z, tak ho zbojníci olúpia o náklad. To znamená, že ak hráč už vyzdvihol svoj materiál, tak na dokončenie misie sa poň bude musieť vrátiť.
- RC-Rozbitá cesta: Ak hráč stúpi na políčko s týmto negatívnym žetónom a taktiež je iného hráča, tak putuje do väzenia.
- B-Bažina: Ak hráč vstúpil na políčko, na ktorom je cudzí žetón B, tak sa vracia na políčko odkiaľ prišiel.

Pozitívne žetóny

Hráč si pozitívne žetóny môže kúpiť na svojom ťahu. Ich funkcionalitou je brániť sa voči negatívnym žetónom. Ku každému negatívnemu existuje pozitívny, ktorý slúži na ubránenie voči negatívnemu. Konkrétne:

- Ochranka: ochrana pred zbojníkmi

- Nové koleso: imunita voči rozbitej ceste
- Sprievodca: imunita voči bažine

V prípade, že hráč má nakúpený správny (napríklad stúpil na políčko s bažinou a má sprievodcu) pozitívny žetón a zároveň stúpi na políčko, na ktorom je cudzí negatívny, tak sa automaticky vyhne efektu negatívneho žetónu a svoj pozitívny žetón stráca. Následne sa negatívny žetón odstráni z políčka.

Peniaze

Platidlom v tejto hre je mena koruna. Každý negatívny žetón stojí 1 korunu a každý pozitívny žetón stojí 2 koruny. Hráč začína s desiatimi korunami a za každú ďalšiu splnenú misiu dostane toľko korún, koľko je odmena danej misie.

Väzenie

Ak hráč stúpil na poličko, na ktorom je bažina cudzieho hráča a zároveň nemá pozitívny žetón sprievodcu, tak putuje do väzenia. Automaticky končí svoj ťah, teda nemôže nakupovať žetóny ani vkladať negatívne na hraciu plochu a svoj ťah začína protihráč, ktorý po skončení ťahu nasleduje ešte raz. Následne ak ukončí ťah druhýkrát po sebe tak nasleduje hráč, ktorý bol vo väzení. V prípade, že hráč A na svojom ťaju stúpil na bažinu a nemá sprievodcu a následne hráč B na svojom ťahu putuje do väzenia, tak zjavne nepôjde dvakrát po sebe, ale hneď ukončí svoj ťah a nasleduje normálne hráč A a obaja sú z väzenia von.

2.3 Doplňujúce informácie ku hre

Každému hráčovi je na začiatku hry náhodne pridaná misia a figúrka sa postaví na hrad/mesto, v ktorom misia začína. Na splnenie prvej misie teda stačí len materiál doniesť do cieľového hradu/mesta. Ak hráč vyzdvihol tovar a jeho figúrka sa nachádza v cieľovej destinácii misie, tak automaticky získava toľko bodov a toľko peňazí aká je odmena misie a daný hráč dostane náhodne ďalšiu misiu na splnenie. Hráč môže použiť svoj kúpený negatívny žetón v ktoromkoľvek svojom ťahu, teda nie je povinný ho použiť vtedy, kedy ho aj kúpil. Na konkrétnom políčku hracej plochy môžu byť v jednom momente figúrky všetkých hráčov.
Koniec hry: Ak niektorý z hráčov získa 20 a viac bodov, tak hra sa automaticky ukončí a víťazom sa stáva hráč, ktorý tento limit dosiahol.

Materiál	**Z hradu/mesta**	**Do hradu/mesta**	**Počet políčok**	**Odmena**
Železo	Šarišský hrad	Fiľakovo	28	5
Koža	Spišský hrad	Bratislava	24	4

Tabuľka 2.1: Ukážka dvoch misií.

Kapitola 3

Implementácia hry v GDL

V tejto kapitole popíšeme aritmetické operácie, ktoré sú potrebné pre správny priebeh spoločenskej hry Slovania obchodníkmi a zároveň si ukážeme konkrétnu implementáciu hry v GDL jazyku.

3.1 Aritmetické operácie

Narozdiel od iných programovacích jazykov, kde čísla a základné aritmetické operácie sú samozrejmosťou, GDL je jedinečný v tom, že ich nepozná a je nutné si ich dodefinovať. Práve preto ešte pred písaním konkrétnych pravidiel týkajúcich sa hry si musíme zaviesť tieto operácie.

3.1.1 Číslo

V našej hre budeme pracovať len s prirodzenými číslami s 0. Ich základnou vlastnosťou je následnosť. Definujme teda predikát nasledovník (succ z anglického succed) nasledovne:

```
(succ 0 1)
(succ 1 2)
```

Čiže 1 je nasledovníkom 0, 2 je nasledovníkom 1 a tak ďalej. Predikát *succ* budeme v našej práci veľmi často využívať. Je dobrým nástrojom na rýchle zmenšenie alebo zväčšenie čísla o 1. Tatkiež tento predikát bude výborným nástrojom pri definovaní ďalších operácií ako sčítavanie alebo väčší než. Definícia čísla x potom vyzerá takto:

```
(<= (number ?x) (Succ ?y ?x))
(<= (number ?x) (Succ ?x ?y))
```

Alebo je x nasledovníkom y alebo y je nasledovníkom x.

3.1.2 Sčítavanie

Sčítavanie patrí k najzákladnejším operáciam v programovacích jazykoch a nevyhneme sa mu aj v našej práci. Pri definovaní predikátu *pluss* si vystačíme s operáciou nasledovníka a jednoduchou rekurziou. Prvá časť definuje triviálny prípad, pričom $x+0 = x$. Druhá časť reprezentuje $x + y = result$. Nakoľko vieme, že operácia sčítania je komutatívna, tak môžeme napríkad x zväčšiť o 1 a y zmenšiť o 1 a následne sa rekurzívne zavolať. Tým dosiahneme, že sa y postupne rekurziou dostane až na 0 a následne postupným vynorením z rekurzie dostaneme správny výsledok.

```
(<= (pluss ?res 0 ?res)
    (succ ?res ?x))

(<= (pluss ?x ?y ?res)
    (succ ?x ?xP1)
    (succ ?yM1 ?y)
    (pluss ?xP1 ?yM1 ?res))
```

3.1.3 Odčítavanie

V našej hre si vystačíme s operáciou odčítavania, teda $x - y = result$, za podmienky, že $x \geq y$. Potom definícia je takmer identická s operáciou sčítavania.

```
(<= (minuss ?res 0 ?res)     (succ ?res ?x))
(<= (minuss ?x ?y ?res)
    (succ ?xM1 ?x)
    (succ ?yM1 ?y)
    (minuss ?xM1 ?yM1 ?res))
```

3.1.4 Väčší než

Ak už máme funkčný predikát plus, následne pomocou neho môžeme jednoducho zapísať operáciu väčší než, teda x je väčšie ako y, ak existuje z rôzne od nuly a zároveň platí $y + z = x$.

```
(<= (greaterThen ?x ?y)
(pluss ?y ?z ?x)
(not (succ ?z 1)))
```

3.2 Implementácia hry

Ak už máme všetky potrebné operácie pripravené, môžeme začať s písanim pravidiel samotnej hry. Hra je určená pre dvoch hráčov. Prvého hráča budeme názývať *blue* a druhého *red*. Prvky náhody, ktoré hra obsahuje budeme reprezentovať tretím hráčom, nazývaným *random*. Pomocou jeho legálnych ťahov budeme pridávať do hry náhodne prvky. Konkrétna implementácia rolí bude vyzerať triviálne takto:

```
(role blue)
(role red)
(role random).
```

3.2.1 Fakty

Zadefinujme si najprv základné fakty, ktoré budeme v implementácií mnohokrát využívať. Tieto fakty sú počas priebehu hry nemeniace a vždy pravdivé. Ako sme spomínali v kapitole Spoločenská hra "Slovania obchodníkmi"(asi odkaz na nu) TODO, tak hracia plocha sa skladá z políčok ciest a hradov, ktoré pre jednoduchosť budeme identifikovať číslom, teda každé políčko bude mať svoje špecifické identifikačné číslo (označujeme ID). Už samotná hra je oproti ostatným hrám napísaným v GDL obšírna, rozhodol som sa hraciu plochu od tej skutočnej zjednodušiť a zmenšiť, aby sme skrátili dĺžku trvania hry, teda počtu všetkých stavov danej konfigurácie hry z počiatočného do koncového stavu. Hracia plocha bude neohodnotený graf, pričom vrcholy budú políčka hradov a ciest, ktoré potrebujeme od seba odlíšiť, teda konkrétne:

```
;; hrady
(castle 1) (castle 3) (castle 7) (castle 9)
;; cesty
(path 2) (path 4) (path 5) (path 6) (path 8)
```

Ak vedie hrana z políčka A do políčka B, tak automaticky vedie hrana z B do A, preto by sme mohli hraciu plochu reprezentovať neorientovaným grafom, no napriek

tomu pre zjednodušenie písania pravidiel som sa rozhodol faktom $edge(idFrom, idTo)$ reprezentovať len hranu z $idFrom$ do $idTo$. Pomocou tohto faktu následne budeme reprezentovať hrany v orientovanom grafe.

```
(edge 1 2) (edge 2 1) (edge 2 5) (edge 5 2) (edge 5 6) (edge 6 5)
(edge 6 7) (edge 7 6) (edge 5 8) (edge 8 5) (edge 8 9) (edge 9 8)
(edge 4 5) (edge 5 4) (edge 3 4) (edge 4 3)
```

Pre lepšiu predstavu nám pomôže jednoduchá vizualizácia hracieho plánu a to na obrázku x (vytvoriť obr). TODO
Cieľom oboch hráčov je získať 20 TODO (ešte neviem presne aký bude limit, či 10 či 20) bodov tým, že plnia misie, ktoré sú im pridané náhodne. Misia sa skladá z identifikačného čísla ID, z hradu/mesta kde ma hráč tovar vyzdvihnúť a z hradu/mesta kam tento tovar má doniesť. V našom prípade množina misií sa skladá zo všetkých kombinácií hradov a miest, avšak začiatočný hrad sa musí líšiť od toho cieľového:

```
(mission 1 1 3) (mission 2 3 7) (mission 3 1 9) (mission 4 7 1)
(mission 5 7 3) (mission 6 7 9) (mission 7 3 1) (mission 8 3 7)
(mission 9 3 9) (mission 10 9 7) (mission 11 9 1) (mission 12 9 3)
```

3.2.2 Počiatočný stav a generovanie hry

Aby Všeobecný Počítačový Hráč bol schopný hrať hocijakú hru, tak je nutnou podmienkou aby aspoň jeden zo stavov hry bol počiatočný. Explicitne inicializujeme všetky dôležité hodnoty hráčom týmto spôsobom:

```
(init (generateGame true))
(init (pozitiveChips blue 0 0 0)) ; meno sprievodca ochranka noveKoleso
(init (pozitiveChips red 0 0 0))
(init (negativeChips blue 0 0 0)) ; meno bazina zbojnici rozbitaCesta
(init (negativeChips red 0 0 0))
```

Pri pozitívnych sú atribútmi postupne meno hráča, počet sprievodcov, ochraniek a nových kolies a u negatívnych v slede meno hráča, počet bažín, zbojníkov a rozbitých ciest. Spoločenská hra Slovania obchodníkmi umožňuje zahrávať akcie len hráčovi, ktorý je na ťahu. Teda ak je na ťahu napríklad hráč *red*, tak potom *blue* a *random* nerobia nič (označujeme noop z anglického no operiaton). Na vygenerovanie hry budeme potrebovať pridať misie hráčom náhodne a to zabezpečíme pomocou legálneho ťahu hráča *random*. Legálne ťahy pri generovaní vyzerajú potom nasledovne:

```
;hraci okrem random nerobia nic pri generovani hry
(<= (legal ?anyPlayer noop)
    (true (generateGame true))
    (role ?anyPlayer)
    (distinct ?anyPlayer random))

;vsetky mozne kombinacie misii, z ktorych random nasledne vyberie nahodne
(<= (legal random (generateMissions ?idBlueM ?idRedM))
    (true (generateGame true))
    (mission ?idBlueM ?from ?to)
    (mission ?idRedM ?from2 ?to2)
    (distinct ?idBlueM ?idRedM))
```

Prvé pravidlo nám hovorí, že ak sme v stave, kedy generujeme hru, tak hráči *blue* a *red* nerobia nič a druhé pravidlo slúži práve na vygenerovanie misií pre hráčov, teda *random* bude môcť vybrať zo všetkých kombinácií faktov misií, avšak nemôžu mať obaja hráči rovnaké ID misie. Atribúty v $generateMissions$ sú v poradí prvý pre *blue* a druhý pre *red* hráča. Na reprezentovanie všetkých dôležitých atribútov hráča budeme používať predikát *playerStat*, ktorého atribúty sú v tomto poradí, hráč, pozícia figúrky, peniaze, ID misie, dosiahnuté body a či už vyzdvihol tovar. Napríklad konkrétny predikát môže vyzerať takto: $playerStat(blue, 1, 10, 5, 6, false)$ Aby sme generovanie úspešne dokončili, potrebujeme ešte definovať pomocou relácie *next*, čo bude platiť v nasledujúcom stave. Inicializovanie pre hráča *blue* vyzerá takto:

```
(<= (next (playerStat blue ?from 10 ?idBlueM 1 true))
    (does random (generateMissions ?idBlueM ?idRedM))
    (mission ?idBlueM ?from ?to))
```

Hráč *blue* dostane id misie podľa toho, čo vygeneroval hráč *random* a podľa faktu misie pridáme aj aktuálnu pozíciu figúrky, ktorá je totožná so začiatkom misie a zároveň môžeme pridať *true* pre atribút $vyzdviholTovar$ a práve preto inicializujeme aktuálny počet bodov na 1 (z pravidiel vieme, že ak hráč vyzdvihne tovar tak jeho aktuálny počet bodov sa zvíši o 1). Na dokončenie všetkých atribútov predikátu *playerStat* pridáme ešte 10 korún. Inicializovanie pre hráča *red* je analogické, len ID misie je v poradí druhý atribút v $generateMissions$, čiže:

```
(<= (next (playerStat red ?from 10 ?idRedM 1 true))
    (does random (generateMissions ?idBlueM ?idRedM))
```

```
    (mission ?idRedM ?from ?to))
```

Na reprezentovanie informácie, ktorý hráč je práve na ťahu a kto bude po ňom nasledovať bude slúžiť predikát $control(aktualnyHrac, nasledujuciHrac)$. Každému stavu, kedy bude môcť *blue* alebo *red* hrať svoje akcie predchádza ťah hráča *random*, ktorý vygeneruje číslo na kocke, o ktoré sa následne hráč bude môcť posunúť. Preto potrebujeme zadefinovať, ktorý hráč bude na ťahu v nasledujúcom stave od počiatočného a to zabezpečíme týmito pravidlami:

```
(<= (next (control random blue))
    (does random (generateMissions ?idBlueM ?idRedM))
    (greaterThen ?idBlueM ?idRedM))

(<= (next (control random red))
    (does random (generateMissions ?idBlueM ?idRedM))
    (greaterThen ?idRedM ?idBlueM))
```

Teda v budúcom stave bude na ťahu hráč *random*, aby vygeneroval číslo hodené na kocke a po ňom bude nasledovať ten hráč, ktorý dostal vyššie id misie, čo je v podstate analógia s inými spoločenskými hrami, kde začína hráč, ktorý hodil vyššie číslo na kocke. Na lepšiu predstavu simulácie generovania hry nám pomôže obrázok 3.1.

3.2.3 Legálne ťahy hráčov

Ak sme už novú hru úspešne vygenerovali, tak si môžeme postupne vysvetliť všetky legálne akcie hráčov. Ukážme si najprv ten najzákladnejší a to ak hráč nie je na ťahu, tak nerobí nič:

```
;hraci mimo ťahu nerobia nic
(<= (legal ?anyPlayer noop)
    (true (control ?player ?other))
    (role ?anyPlayer)
    (distinct ?player ?anyPlayer))
```

Základný prvok náhody, ktorý využívame v hre pred začiatkom ťahu hráčov je hod kockou a zabezpečíme ho pomocou hráča *random*. Pravidlo pre hod čísla 1 vyzerá potom takto:

```
;ak sa este hrac nepohol, tak mozeme hadzat kockou pomocou random hraca,
```

```
mozne hody, zatial 1 a 2
(<= (legal random (roll 1))
    (true (control random ?next))
    (not (true (moved ?next)))
    (not (true (generateGame true))))
```

Čiže ak platí, že hráč *random* je na ťahu, nasledujúci hráč sa ešte nepohol (táto podmienka je nutná, pretože ak hráč dokončí misiu a *random* mu následne pridá novú, tak nechceme aby mal na výber hodiť číslo) a zároveň hra už je vygenerovaná. Kocku sme si v našej hre prispôsobili a to tak, že má len hodnoty 1,2,3 a 4, nakoľko sme si hraciu plochu zmenšili. Implementácia hodenia kocky 2,3 a 4 je analogická ako vyššie s tým rozdielom že v roll namiesto 1 by bolo 2,3 alebo 4. Legálne akcie mimo ťahu sme už popísali a vysvetlili, teraz sa zamerajme na legálne akcie hráčov na ťahu a rozdeľme ch na 3 časti:

1. posun figúrkou
2. nákup negatívnych a pozitívnych žetónov, vkladanie negatívnych žetónov na políčka ciest
3. ukončenie ťahu

Začnime prvým bodom, ktorý je nevyhnutný, čiže každý hráč začína svoj ťah posunom figúrky o číslo, ktoré vygeneroval *random*. Ako zistíme aké číslo hodil *random* na kocke? Pomocou predikátu *rollRess*, ktorý získame pomocou relácie *next*:

```
(<= (next (rollRess ?num))
    (does random (roll ?num)))
```

Ak *random* hodil číslo, tak v nasledujúcom stave bude platiť predikát *rollRess* s rovnakým číslom. Je nevyhnutné aby to bolo takto definované cez nový predikát a reláciu *next*, pretože GDL nepodporuje závislosť relácie *legal* od relácie *does*. Akcia zahrania posunu figúrky vyzerá nasledovne:

```
(<= (legal ?player (move ?act ?next)) ;pohyb figurky hraca o hodene cislo
    (true (rollRess ?num))
    (or (vrcholyVoVzdialenosti ?act ?next ?num)
    (hradyVoVzdialenosti ?act ?next ?num))
    (true (playerStat ?player ?act ?m ?idM ?p ?collected))
    (true (control ?player random)))
```

Pravidlo nám hovorí, že hráč, ktorý je na ťahu môže pohnúť figúrku z aktuálnej pozície na nové políčko, o ktorom musí platiť aspoň jedna z týchto podmienok:

- Ak random hodil číslo x, tak nové políčko (cesta) musí byť vo vzdialenosti presne x od aktuálnej pozícii figúrky.
- Ak random hodil číslo x, tak nové políčko (hrad) musí byť vo vdzialenosti maximálne x.

Prvá podmienka závisi od pomocného predikátu ktorý vyzerá takto:

```
(<= (vrcholyVoVzdialenosti ?a ?b 1)
    (edge ?a ?b))

(<= (vrcholyVoVzdialenosti ?a ?b ?n)
    (greaterThen ?n 1)
    (edge ?a ?c)
    (succ ?x ?b)
    (succ ?nM1 ?n)
    (not (equals ?a ?b))
    (vrcholyVoVzdialenosti ?c ?b ?nM1))
```

Prvá časť definuje triviálny prípad, kedy vrchol A je vo vzdialenosti presne 1 od vrcholu B, ak existuje hrana z A do B. Druhá časť nám hovorí o tom, že vzdialenosť vrcholu A od vrcholu B sa rovná n ak platí, že vrcholy A a B sú odlišné, vzdialenosť n je väčsia ako 1 a zároveň existuje vrchol C, ktorý je vo vzdialenosti od B presne $n-1$ a to vypočítame pomocou jednoduchej rekurzie.
Druhá podmienka závisí od následujúceho doplnkového pravidla:

```
;vsetky hrady vo vzdialenosti "n", ale v skutocnosti su vo vzdialenosti mensej
ako n a vacsej ako 0
(<= (hradyVoVzdialenosti ?a ?cas ?n)
    (castle ?cas)
    (greaterThen ?x 0)
    (greaterThen ?n ?x)
    (vrcholyVoVzdialenosti ?a ?cas ?x))
```

Hrad je vo vzdialenosti n od vrcholu A ak platí, že hrad je skutočne hradom a zároveň je vo vzdialenosti 1 až $n-1$ od vrcholu A. Táto vlastnosť pôsobí mätúco, ale má svoj význam. Ak sa napríklad nachádzame na políčku, ktoré je vo vzdialenosti od

hradu kam máme doniesť tovar misie presne 2 a na kocke hodíme číslo 4, tak práve toto pravidlo nám slúži na to aby sme mali možnosť pohnúť sa s figúrkou aj na tento hrad. Na lepšie pochopenie môžeme tento legálny ťah demonštrovať na obrázku x. (vytvorit obrázok. jednak kde budú predikáty aktuálneho stavu a možnosti kam hráč sa môže pohnúť, plus jednoduchý obrázok hracej plochy s vyraznenými políčkami kam môže vsúpiť.) TODO

Prejdime teraz do druhej časti legálnych akcií hráča na ťahu. Hráč môže nakúpiť negatívne žetóny a to bažinu, zbojníkov a rozbitú cestu. Na všetky tri platia rovnaké podmienky, preto si ukážeme len nákup bažiny a analogicky je potom definovaný aj nákup zbojníkov či rozbitej cesty, len s inými názvami, konkrétne *buyN_Zbojnici* a *buyN_RozbitaCesta*.

```
(<= (legal ?player buyN_Bazina)
    (true (playerStat ?player ?act ?m ?idM ?p ?collected))
    (true (control ?player random))
    (true (moved ?player))
    (not (true (inJail ?player)))
    (greaterThen ?m 0))
```

Hráč môže nakúpiť negatívny žetón s názvom bažina ak je na ťahu, už sa pohol figúrkou, nie je vo väzení a má dostatok peňazí, konkrétne negatívne žetóny stoja jednu korunu za jeden kus, preto stačí ak má počet peňazí vyšší ako 0. Aby sme pochopili do detailov toto pravidlo, tak si musíme ešte ukázať, kedy je pravdivý predikát *moved*(*player*):

```
(<= (next (moved ?player))
    (does ?player (move ?from ?to)))

(<= (next (moved ?player))
    (true (control ?player random))
    (not (does ?player endTurn)))

(<= (next (moved ?player))
    (does random (generateNewMission ?idM))
    (true (control random ?player)))
```

Toto pravidlo nám slúži na to aby sme vedeli kedy hráč môže zahrať aj iné ťahy ako len *move*. Prvá časť slúži na vytorenie predikát, teda zjavne v nasledujúcom stave bude platiť moved, ak hráč práve zahral akciu *move*. Druhá časť nám slúži na zotrvanie

predikátu, teda ak hráč je na ťahu a ešte neukončil ťah tak predikát sa nemení. Tretia časť je pridaná pre situáciu, ak hráč dokončil misiu a následne budeme potrebovať pomocou hráča *random* vygenerovať novú misiu, tak aby po vygenerovaní stále mohol pokračovať v hraní legálnych akcií. Pre úplne pochopenie si ešte ukážeme, kedy platí predikát *inJail*. Pravidlo sa skladá z dvoch častí, tá prvá slúži na vytvorenie predikátu a vyzerá nasledovne:

```
(<= (next (inJail ?player))
    (true (field ?newP ?opponent rozbitaCesta))
    (does ?player (move ?oldP ?newP))
    (true (pozitiveChips ?player ?s ?o 0))
    (distinct ?player ?opponent))
```

Hráč bude v nasledujúcom stave vo väzení ak vstúpil na políčko, na ktorom je súperov negatívny žetón *rozbitaCesta* a zároveň nemá pozitívny žetón *noveKoleso* (ochranu voči rozbitej ceste). Druhá časť slúži na kopírovanie predikátu v nezmenenej podobe:

```
(<= (next (inJail ?player))
    (true (inJail ?player))
    (not (removeJail ?player)))
```

Teda jednoducho hráč ostáva vo väzení ak v ňom je a zároveň nenastane možnosť, aby sa z neho dostal preč. Prípad kedy sa hráč má dostať von z väzenia je definovaný týmto spôsobom:

```
(<= (removeJail ?player)
    (does ?opponent endTurn)
    (role ?player)
    (distinct ?player ?opponent))

(<= (removeJail ?player)
    (true (inJail ?player))
    (true (inJail ?opponent))
    (distinct ?player ?opponent))
```

Hráč sa dostáva z väzenia ak ukončil ťah súper alebo ak je vo väzení aj súper.
Hráč môže taktiež nakúpiť pozitívne žetóny aby sa dokázal brániť voči negatívnym. Môže si vybrať z týchto žetónov a to sprievodcu, ochranku a nové koleso. Už z názvov je zrejmé, že sprievodca slúži na obranu voči bažine, ochranka na zbojníkov a nové

koleso na rozbitú cestu. Nákup ochranky je podobný nákupu bažiny a to:

```
(<= (legal ?player buyP_Ochranka)
    (true (control ?player random))
    (not (true (inJail ?player)))
    (true (playerStat ?player ?act ?m ?idM ?p ?collected))
    (true (moved ?player))
    (greaterThen ?m 1))
```

Hráč môže zahrať legálnu akciu nákupu ochranky ak je na ťahu, pohol sa, nie je vo väzení a zároveň jeho počet peňazí je väčší ako 1, nakoľko pozitívne žetóny stoja 2 koruny za jeden kus. Analogicky sú zadefinované nákupy zbojníkov a nových kolies len s iným názvom. Negatívne žetóny môžu hráči na svojom ťahu vkladať na hraciu plochu, tak ukážeme si práve pravidlo, ktoré reprezentuje legálnu akciu vloženia bažiny na políčko:

```
(<= (legal ?player (dropBazina ?policko))
    (path ?policko)
    (not (isField ?policko))
    (true (control ?player random))
    (true (moved ?player))
    (role ?anyPlayer)
    (not (true (inJail ?player)))
    (true (negativeChips ?player ?b ?z ?rc))
    (greaterThen ?b 0))
```

Hráč môže vložiť bažinu na políčko ak políčko je voľná cesta, teda neobsahuje iný negatívny žetón a zároveň platí, že hráč je na ťahu, už pohol figúrkou, nie je vo väzení a má práve nejakú bažinu kúpenú, teda počet bažín je väčší ako 0. Analogicky sú definované legálne akcie vloženia zbojníkov, avšak s tým rozdielom, že počet práve nakúpených zbojníkov je väčší ako 0 a podobne u legálnej akcie vloženia rozbitej cesty počet nakúpených rozbitých ciest musí byť väčší ako 0. Pre úplne pochopenie si špecifikujme predikát *isField*.

```
(<= (isField ?policko)
    (true (field ?policko ?player ?chip)))
```

Teda políčko je obsadené ak platí že na tom políčku už je nejaký negatívny žetón nejakého hráča. Kedy však platí predikát *field*? Máme tri pravidlá, ktoré zabezpečujú

vytvorenie predikátu *field* v nasledujúcom stave. Ukážeme si jedno z nich:

```
(<= (next (field ?policko ?player bazina))
    (does ?player (dropBazina ?policko)))
```

Na políčku bude v nasledujúcom stave bažina ak hráč zahral legálnu akciu vloženia bažiny na políčko. Analogicky na políčku v budúcom stave budú zbojníci, respektíve rozbitá cesta, ak hráč zahral *dropZbojnici*, respektíve *drobRozbitaCesta*. Nasledujúce pravidlo slúži na zachovanie predikátu v identickej podobe.

```
(<= (next (field ?f ?player ?chip))
    (true (field ?f ?player ?chip))
    (not (removeField ?f ?player ?chip)))
```

Žetón ostane na políčku v nastavajúcom stave, ak tam už je a neplatí, že má zmiznúť. Ukážeme si najprv kedy bažina a rozbitá cesta sa má odstrániť z políčka:

```
(<= (removeField ?newP ?playerPut ?chip)
    (distinct ?chip zbojnici)
    (true (field ?newP ?playerPut ?chip))
    (does ?otherPlayer (move ?old ?newP))
    (distinct ?otherPlayer ?playerPut))
```

Ak negatívny žetón je odlišný od zbojníkov a súčasne je položený na políčku a zároveň iný hráč stúpil na dané políčko. Odstránenie zbojníkov je analogické avšak s pridaním jednej podmienky a tou je, že zbojníci okrádajú len už vyzdvihnutý tovar, teda ak chceme aby zmizli, tak cudzí hráč, ktorý stúpi na políčko musí už mať tovar vyzdvihnutý. Konkrétna implementácia vyzerá nasledovne:

```
(<= (removeField ?newP ?playerPut zbojnici)
    (true (field ?newP ?playerPut zbojnici))
    (does ?otherPlayer (move ?old ?newP))
    (true (playerStat ?otherplayer ?old ?m ?idM ?p true))
    (distinct ?otherPlayer ?playerPut))
```

Predposledným legálnym ťahom v hre Slovania obchodníkmi je generovanie novej misie:

```
(<= (legal random (generateNewMission ?idM))
    (true (finished_missionn ?player))
```

```
    (mission ?idM ?from ?to))
```

Hráč random môže zahrať akciu generovania novej misie ak platí, že nejaký hráč dokončil misiu a zároveň existuje misia s daným ID. Pomocný predikát *finished_missionn* vyzerá takto:

```
(<= (next (finished_missionn ?player))
    (does ?player (move ?act ?newP))
    (true (playerStat ?player ?act ?m ?actMission ?actPoints true))
    (mission ?actMission ?from ?newP))
```

Hráč dokončil misiu ak sa pohol na políčko, ktoré sa rovná políčku cieľového hradu aktuálnej misie.
Poslednou legálnou akciou je ukončenie ťahu. Hráč môže ukončiť svoj ťah ak sa pohol a zároveň je na ťahu.

```
(<= (legal ?player endTurn)
    (true (moved ?player))
    (true (control ?player random)))
```

3.2.4 Následujúci stav

Každá hra napísaná v GDL využíva reláciu *next*, ktorá nám hovorí, čo bude platiť v nasledujúcom stave. Nejaké pomocné pravidlá využívajúce *next* sme si už ukázali v predošlých podkapitolách, avšak nedefinovali sme ešte základné predikáty, ktoré musia platiť pre oboch hráčoch v každom stave a to *negativeChips*, *pozitiveChips* a *playerStat*. Prvý z nich slúži na reprezentáciu počtu nakúpených určitých negatívnych žetónov konkrétnym hráčom. V počiatočnom stave sme si inicializovali počty bažín, zbojníkov a rozbitých ciest na 0. Ukážeme si najprv pravidlá, ktoré hovoria o zmene predikátu z aktuálneho do nasledujúceho stavu. Príklad budeme demonštrovať na žetóne bažina:

```
(<= (next (negativeChips ?player ?newB ?z ?rc)) ;ak kupil bazinu
    (true (negativeChips ?player ?oldB ?z ?rc))
    (does ?player buyN_Bazina)
    (succ ?oldB ?newB))

(<= (next (negativeChips ?player ?newB ?z ?rc)) ;ak dropol bazinu
    (true (negativeChips ?player ?oldB ?z ?rc))
```

```
    (does ?player (dropBazina ?policko))
    (succ ?newB ?oldB))
```

To jest počet bažín, ktoré hráč vlastní sa zvýši o jedna, ak žetón kúpil a zníži, ak ho položil na políčko hracej plochy. Analogicky u zvyšných dvoch žetónoch sa podobne počet zvýši o jedna, ak hráč kúpil daný žetón a zníži, ak položil daný žetón. Potrebujeme ešte zadefinovať pravidlo, ktoré hovorí o tom, že predikát ostane v nezmenenom stave:

```
(<= (next (negativeChips ?player ?b ?z ?rc))
    (true (negativeChips ?player ?b ?z ?rc))
    (not (buyNegativeChip ?player))
    (not (droppedNegativeChip ?player)))
```

Teda počet negatívnych žetónov sa nezmení ak hráč nekúpil ani jeden z negatívnych žetónov a zároveň ani jeden z nich nepoložil na hraciu plochu. Pomocou predikátu pozitiveChips reprezentujeme jednotlivo aktuálny počet všetkých troch pozitívnych žetónov daného hráča v poradí sprievodca, ochranka a nové koleso. Tie slúžia hráčom na obranu voči negatívnym, konkrétne v takýchto dvojiciach, sprievodca na bažinu, ochranka na zbojníkov a nové koleso na rozbitú cestu. Analogicky ako u negatívnych sa počet daného pozitívneho žetóna zvýši o jedna ak hráč zahral legálnu akciu nákupu daného žetóna. Na druhej strane zníženie počtu o jedna je odlišné a nastane vtedy, ak pozitívny žetón splní svoju funkciu. Pravidlo si ukážeme na pozitívnom žetóne sprievodca.

```
(<= (next (pozitiveChips ?player ?sM1 ?o ?nk)) ;ak sprievodca splnil svoju
funkciu
    (stepOnBazinaAndHaveSprievodca ?player)
     (true (pozitiveChips ?player ?actS ?o ?nk))
    (greaterThen ?actS 0)
    (succ ?sM1 ?actS))
```

Čiže počet sprievodcov sa zníži o jeden ak hráč má aspoň jedného sprievodcu a platí predikát stepOnBazinaAndHaveSprievodca , ktorý vyzerá nasledovne:

```
(<= (stepOnBazinaAndHaveSprievodca ?player)
    (does ?player (move ?oldP ?newP))
    (true (field ?newP ?opponent bazina))
    (distinct ?player ?opponent)
```

```
    (true (pozitiveChips ?player ?s ?o ?nk))
    (greaterThen ?s 0))
```

Teda predikát je pravdivý, ak hráč vstúpil figúrkou na políčko, na ktorom je cudzia bažina. Analogicky sa počet nových kolies, respektíve ochraniek zníži o jedna ak hráč vstúpil na cudziu rozbitú cestu, respektíve zbojníkov a zároveň hráč už mal vyzdvihnutý tovar, pretože len vtedy môžu zbojníci olúpiť hráča o náklad. Počet pozitívnych žetónov hráča ostane v budúcom stave rovnaký ak nekúpil ani jeden z pozitívnych žetónov a zároveň ani jeden z nich nesplnil svoju funkciu. Pri generovaní hry sme inicializovali predikát playerStat, teraz si ukážeme pár pravidiel s reláciou next, ktoré nám hovoria buď o zmene tohto predikátu v budúcom stave alebo o jeho zachovaní v nezmenenom tvare. Napríklad toto pravidlo:

```
(<= (next (playerStat ?player ?actP ?m ?idM ?p ?collected)) ;ak hrac nie
je na tahu, figurka ostava na rovnakom poli hracej plochy
    (does ?player noop)
    (not (true (finished_missionn ?player)))
    (true (playerStat ?player ?actP ?m ?idM ?p ?collected)))
```

Ak hráč nie je na ťahu, tak stav hráča ostáva nezmenený, čiže všetky jeho dôležité atribúty ostávajú rovnaké. Ďalším pravidlom demonštrujúcim zhodný predikát v nasledujúcom stave je:

```
(<= (next (playerStat ?player ?act ?m ?idM ?p ?collected)) ;ak ukoncil tah
alebo dropol negativny zeton
    (true (playerStat ?player ?act ?m ?idM ?p ?collected))
    (or (does ?player endTurn) (does ?player (dropBazina ?policko)) (does
?player (dropZbojnici ?policko)) )
```

Ak hráč ukončil svoj ťah alebo položil na stôl negatívny žetón, tak sa nemení žiaden atribút z nasledovných: aktuálna pozícia, počet peňazí, id misie, dosiahnuté body, či atribút reprezentujúci vyzdvihnutie tovaru. Znázornime si teraz nejaké prípady zmeny predikátu playerStat, napríklad:

```
(<= (next (playerStat ?player ?newP ?m ?actMission ?newPoints true)) ;ak
je na policku kde zacina misia tak vyzdivhol tovar
    (true (playerStat ?player ?act ?m ?actMission ?p false))
    (succ ?p ?newPoints)
```

```
    (does ?player (move ?act ?newP))
    (mission ?actMission ?newP ?to))
```

Môžeme si všimnúť, že sa zmení atribút jednak pozície figúrky kam sa hráč pohol, ale taktiež aj počet dosiahnutých bodov sa zvýši o 1, pretože už vyzdvihol tovar a zároveň atribút indikujúci vyzdvihnutie tovaru sa zmení na true. Kvázi jednoduché zmeny nastanú ak hráč nakúpi nejaký negatívny žetón, tak zmenší sa o jedna len atribút počtu peňazí, podobne ak nakúpi hráč pozitívny žetón tak sa zmenší počet peňazí o 2, pretože pozitívne žetóy stoja 2 koruny za jeden kus. Ďalší prípad je ak hráč vstúpi na políčko kde začína misia, tak sa zmení atribút vyzdvihnutia tovaru na true, alebo ak hráč vstúpi na políčko na ktorom je cudzí negatívny žetón bažina, tak sa nepohne ale ostáva na pôvodnom políčku a podobne.
Na záver tejto podkapitoly si ešte ukážeme zopár pravidiel obsahujúcich predikát control, ktorý nám slúži na zastupovanie hráča na ťahu a taktiež hráča, ktorý bude nasledovať po ňom. Nasledujúce pravidlo nám hovorí prípad, kedy sa predikát ostane v budúcom stave v nezmenenom stave:

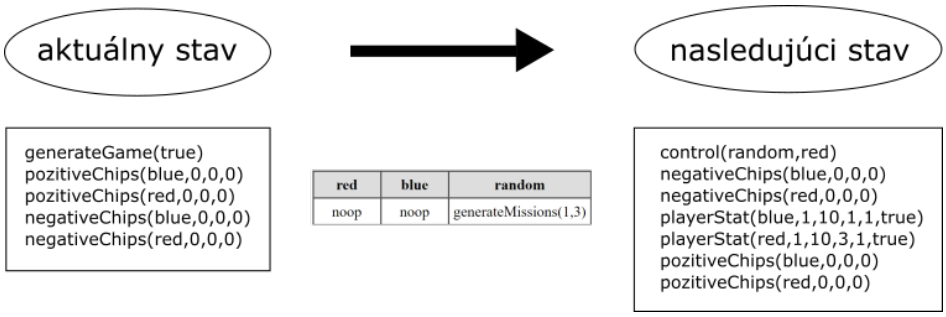
```
(<= (next (control ?player random))
    (true (control ?player random))
    (not (finished_mission ?player))
    (not (does ?player endTurn)))
```

To nastane vtedy, ak hčár neukončil svoj ťah a zároveň nedokončil misiu. Síce je pravda, že ak hráč dokončí misiu tak teoreticky nekončí svoj ťah, ale prakticky potrebujeme aby nasledoval random a vygeneroval novú misiu:

```
(<= (next (control random ?player))
    (finished_mission ?player))

(<= (next (control ?player random))
    (true (control random ?player))
    (does random (generateNewMission ?idM)))
```

Teda v nasledjúcom stave bude môcť vygenerovať misiu random ak hráč ukončil misiu a následne po vygenerovaní bude na ťahu opäť rovnaký hráč.



Obr. 3.1: Na obrázku vidíme konkrétnú ukážku generovania hry, teda prechodu z aktuálneho (počiatočného) stavu do nasledujúceho pomocou legálnych akcií hráčov. Keďže generujeme hru, tak red aj blue nerobia nič a random zahral svoju akciu vygenerovania misií s id 1 pre blue a 3 pre red. V nasledujúcom stave bude na ťahu *random*, aby sme napodobnili hod kocky a po ňom pôjde hráč red, pretože má vyššie ID misie.

3.2.5 Terminálny stav a odmeny

Hra sa dostane do koncového stavu ak niektorý z hráčov má aktuálny počet bodov väčší ako 9:

```
(<= terminal
    (true (playerStat ?player ?actP ?m ?actMission ?actPoints ?collected))
    (greaterThen ?actPoints 9))
```

Pridanie bodov hráčom je triviálne:

```
(<= (goal ?player ?actPoints)
    (true (playerStat ?player ?actP ?m ?actMission ?actPoints ?collected)))
    (goal random 0)
```

Hráč dosiahne na konci hry počet bodov rovný aktuálnemu počtu získaných bodov. A *random* samozrejme dostane 0 bodov.

Kapitola 4

Testovanie počítačových hráčov

Táto kapitola sa zaoberá skúšaním Všeobecných Počítačových Hráčov na už napísanej hre v GDL jazyku.

Záver

Na záver už len odporúčania k samotnej kapitole Záver v bakalárskej práci podľa smernice [**?**]: „V závere je potrebné v stručnosti zhrnúť dosiahnuté výsledky vo vzťahu k stanoveným cieľom. Rozsah záveru je minimálne dve strany. Záver ako kapitola sa nečísluje.“

Všimnite si správne písanie slovenských úvodzoviek okolo predchádzajúceho citátu, ktoré sme dosiahli príkazmi `\glqq` a `\grqq`.

V informatických prácach niekedy býva záver kratší ako dve strany, ale stále by to mal byť rozumne dlhý text, v rozsahu aspoň jednej strany. Okrem dosiahnutých cieľov sa zvyknú rozoberať aj otvorené problémy a námety na ďalšiu prácu v oblasti.

Abstrakt, úvod a záver práce obsahujú podobné informácie. Abstrakt je kratší text, ktorý má pomôcť čitateľovi sa rozhodnúť, či vôbec prácu chce čítať. Úvod má umožniť zorientovať sa v práci skôr než ju začne čítať a záver sumarizuje najdôležitejšie veci po tom, ako prácu prečítal, môže sa teda viac zamerať na detaily a využívať pojmy zavedené v práci.

Literatúra

[1] https://en.wikipedia.org/wiki/Graph_(discrete_mathematics). [Citované 2021-02-03].

[2] https://en.wikipedia.org/wiki/General_game_playing. [Citované 2021-02-10].

[3] Michael Thielscher Michael Genesereth. *General Game Playing (Synthesis Lectures on Artificial Intelligence and Machine Le)*. Morgan & Claypool Publishers, 2014.

[4] Michael Genesereth Nathaniel Love, Timothy Hinrichs. General Game Playing: Game description language specification. Technical report, Stanford University, 2006.

Príloha A: obsah elektronickej prílohy

V elektronickej prílohe priloženej k práci sa nachádza zdrojový kód programu a súbory s výsledkami experimentov. Zdrojový kód je zverejnený aj na stránke `http://mojadresa.com/`.

Ak uznáte za vhodné, môžete tu aj podrobnejšie rozpísať obsah tejto prílohy, prípadne poskytnúť návod na inštaláciu programu. Alternatívou je tieto informácie zahrnúť do samotnej prílohy, alebo ich uviesť na obidvoch miestach.

Príloha B: Používateľská príručka

V tejto prílohe uvádzame používateľskú príručku k nášmu softvéru. Tu by ďalej pokračoval text príručky. V práci nie je potrebné uvádzať používateľskú príručku, pokiaľ je používanie softvéru intuitívne alebo ak výsledkom práce nie je ucelený softvér určený pre používateľov.

V prílohách môžete uviesť aj ďalšie materiály, ktoré by mohli pôsobiť rušivo v hlavnom texte, ako napríklad rozsiahle tabuľky a podobne. Materiály, ktoré sú príliš dlhé na ich tlač, odovzdajte len v electronickej prílohe.