Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Automatický hráč hry Dominion

Bakalárska práca

2012

Peter Fulla

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Automatický hráč hry Dominion

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Michal Forišek, PhD.

Bratislava, 2012

Peter Fulla

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Peter Fulla
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Automatický hráč hry Dominion

Cieľ: Dominion je kartová hra s neúplnou informáciou a viacerými algoritmicky zaujímavými aspektmi. V súčasnosti nie je známa žiadna implementácia počítačového hráča pre túto hru. Cieľom tejto bakalárskej práce je preskúmať možné prístupy vedúce k takejto implementácii a následne implementovať čo najlepšie hrajúceho hráča a otestovať ho v partiách s ľudským protihráčom.

Vedúci: RNDr. Michal Forišek, PhD.
Katedra: FMFI.KI - Katedra informatiky

Dátum zadania: 15.10.2011

Dátum schválenia: 20.10.2011

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Ďakujem MišoFovi,
za skvelú tému práce a rady;

Marike,
za ochotu hrať so mnou Dominion, aj keď by sa práve mala učiť;

Donaldovi X., C. R. Darwinovi, D. E. Knuthovi, mojej rodine, priateľom, známym a všetkým tým, bez ktorých by táto práca nevznikla.

Abstrakt

V tejto práci vytvoríme niekoľko modelov stratégie pre počítačového hráča kartovej hry Dominion, pričom konkrétne hodnoty ich parametrov necháme vyvinúť pomocou evolučných algoritmov. Kvalitu získaných stratégií zhodnotíme hrami s existujúcou referenčnou stratégiou `BigMoney` a s ľudským protihráčom.

Kľúčové slová: Dominion, umelá inteligencia, evolučné algoritmy

Abstract

In this thesis we create several models of strategy for an artificial player of Dominion, a card game. The specific parameters of these models are evolved automatically using an evolutionary algorithm. We measure the quality of obtained strategies in games against a reference strategy (`BigMoney`) and a human player.

Keywords: Dominion, artificial intelligence, evolutionary algorithms

Obsah

Úvod

Umelá inteligencia ako odbor informatiky dosiahla počas svojej krátkej histórie viaceré úspechy v rozličných odvetviach. Niektoré majú zjavné praktické využitie, napríklad rozpoznávanie ľudskej reči alebo robotické systémy. Oproti tomu vyzerá napríklad snaha o vytvorenie počítačového hráča šachu ako neužitočná zábavka.

Stolové hry však predstavujú problémy podobné tým zo skutočného sveta, pričom na ich reprezentáciu používajú len malý počet diskrétnych objektov. Vďaka tomu sa nemusíme zapodievať zložitou manipuláciou s prostredím a analýzou spojitých vnemov – môžeme sa sústrediť na stratégiu a plánovanie našich akcií.

Tvorba automatického hráča tak poskytuje príležitosť na vývoj a testovanie techník, ktoré sú využiteľné aj pri riešení serióznejších úloh.

Cieľom tejto práce je vytvoriť čo najlepšie hrajúceho hráča kartovej hry Dominion. Možné akcie hráčov a teda aj ideálnu stratégiu vo veľkej miere ovplyvňuje to, s ktorými desiatimi druhmi kráľovských kariet sa hráči rozhodnú hrať. Možných variantov hry je však nesmierne veľa. Preto v tejto práci nehľadáme jednu univerzálnu stratégiu – namiesto toho sa zameriame na automatické vytvorenie stratégie podľa zadanej desatice kráľovských kariet.

Na tento účel využijeme evolučné algoritmy. Vytvoríme niekoľko modelov stratégií, pričom ich parametre neurčíme, ale necháme vyvinúť evolučnými procesmi.

V prvej kapitole predstavíme hru Dominion a jej pravidlá. V nasledujúcich dvoch kapitolách popíšeme základné pojmy umelej inteligencie a evolučných algoritmov. Následne popíšeme našu implementáciu prostredia hry Dominion a modely stratégií, ktoré sme použili. V poslednej kapitole uvedieme niekoľko príkladov vyvinutých stratégií spolu so zhodnotením ich kvality.

Kapitola 1

Dominion z pohľadu teórie hier

Hra Dominion nepatrí medzi všeobecne známe, preto ju v tejto kapitole najprv predstavíme a neformálne popíšeme jej pravidlá. Následne použijeme terminológiu z teórie hier a vytvoríme formálny zápis pravidiel Dominionu.

1.1 Najskôr neformálne

Dominion je kartová hra pre dvoch až štyroch hráčov vydaná v roku 2008 spoločnosťou Rio Grande Games. Jej tvorcami sú Donald X. Vaccarino (autor), Valerie Putman a Dale Yu.

Postupne vychádzalo viacero rozšírení Dominionu (Intrigue, Seaside, Alchemy, ...), ktoré priniesli nové typy kariet a rozmanitosť štýlov hry (napríklad rozšírenie Seaside obsahuje karty, ktoré ovplyvňujú nielen aktuálny, ale aj nasledujúci ťah hráča). V tejto práci sa však zaoberáme len základnou sadou kariet.

V ďalšom texte čerpáme najmä z pravidiel [8] dodávaných k základnej sade.

Počas hry si hráči budujú svoje vlastné balíčky kariet, najčastejšie kúpou zo spoločnej zásoby (*supply*). Z každého druhu karty je k dispozícii niekoľko navzájom nerozlíšiteľných kusov. Napríklad na začiatku hry obsahuje spoločná zásoba 40 kariet *Silver*.

Základnými typmi kariet sú:

- karty peňazí (*treasure cards*),
- karty víťazných bodov (*victory cards*) a kliatby (*curse cards*),
- karty akcií (*action cards*).

Karty víťazných bodov a kliatby nemajú počas vlastnej hry nijaké využitie, no po jej skončení rozhodujú o výsledku: Každá karta víťazných bodov v balíku hráča mu prináša nejaký nezáporný počet víťazných bodov (karta *Estate* 1 bod, *Duchy* 3 body, *Province* 6 bodov, *Gardens* 1 bod za každých 10 kariet v balíku hráča). Jediná karta typu kliatba, *Curse*, prináša hráčovi -1 bod. Hru vyhráva ten hráč, ktorý má na konci najvyšší počet víťazných bodov.

Karty peňazí slúžia na kupovanie nových kariet zo spoločnej zásoby. Zvláštnosťou je, že hráč neprichádza o svoje karty peňazí, ktorými pri kúpe platil; môže ich teda neskôr použiť znova. Podrobnejšie popíšeme spôsob kúpy neskôr, spolu s priebehom ťahu hráča.

Karty akcií sú rôznorodého charakteru. Umožňujú hráčovi získať nové karty zo spoločnej zásoby, zbaviť sa neužitočných kariet z vlastného balíka, uškodiť protihráčom, ... Práve širokou variabilitou kariet akcií si Dominion získal srdcia mnohých hráčov.

1.1.1 Možné umiestnenia kariet

Väčšina kariet je na začiatku hry umiestnená v spoločnej zásobe. Tá obsahuje štandardné karty, ktoré sa používajú v každej hre (základné karty peňazí, víťazných bodov a kliatby), a desať druhov kráľovských kariet (*kingdom cards*). V základnej sade sú skoro všetky kráľovské karty kartami akcií (výnimkou sú *Gardens*).

Výber kráľovských kariet, ktoré sa umiestnia do spoločnej zásoby, určuje charakter celej hry – rýchlosť, úroveň interakcie medzi hráčmi aj ich stratégiu. Tvorcovia Dominionu uviedli v pravidlách niekoľko pripravených desatíc kráľovských kariet, ktorými chceli predviesť rôzne zaujímavé kombinácie, no hrať je možné s ľubovoľnými desiatimi druhmi.

Každý hráč má svoju vlastnú ťahaciu kopu (*deck*), odhadzovaciu kopu (*discard pile*) a karty v ruke (*hand cards*).

Karty v ťahacej kope sú uložené lícom nadol, ich názvy teda nikto nepozná; karty v odhadzovacej kope sú zase uložené lícom nahor, takže vrchná karta je známa každému hráčovi. Názvy kariet v ruke vidí len ich vlastník.

Okrem toho existuje ešte jedna kôpka spoločná pre všetkých hráčov – smetisko (*trash pile*). Karty umiestnené na nej sa ďalej v hre nepoužívajú.

1.1.2 Priebeh hry

Na začiatku hry každý z hráčov dostane zo zásoby 7 kariet *Copper* a 3 karty *Estate*, zamieša ich, položí na svoju ťahaciu kopu a potiahne si z nej do ruky päť kariet.

Hráči sa následne striedajú v ťahoch. Každý ťah má tri fázy:

- fáza akcií (*action phase*),
- fáza nákupov (*buy phase*),
- fáza upratovania (*clean-up phase*).

Vo fáze akcií si môže hráč vybrať z ruky jednu kartu akcií, položiť ju na stôl a zahrať ju, t.j. vykonať pokyny, ktoré sú na nej uvedené. Niektoré karty umožňujú po ich zahraní pokračovať ďalšími kartami akcií („pridávajú akcie“).

Fázu nákupov začína hráč vyložením niekoľkých kariet peňazí z ruky na stôl. Jednotlivé druhy kariet peňazí majú rôzne hodnoty (karta *Copper* má hodnotu 1, *Silver* 2 a *Gold* 3) a každá karta má na sebe uvedenú svoju cenu. Hráč si môže zobrať zo zásoby jednu kartu s cenou nepresahujúcou súčet hodnôt kariet peňazí, ktoré vyložil. Získanú kartu si položí na vrch svojej odhadzovacej kopy.

Niektoré karty akcií umožňujú hráčovi zobrať zo zásoby viac kariet („pridávajú nákupy“) alebo drahšie karty („pridávajú peniaze“ alebo dočasne znižujú cenu kariet). Celková cena získaných kariet však stále nesmie presiahnuť súčet hodnôt vyložených kariet peňazí a bonusových peňazí z vykonaných akcií.

Vo fáze upratovania hráč presunie všetky vyložené karty (zahrané karty akcií a karty peňazí) a tiež všetky zostávajúce karty v ruke do odhadzovacej kôpky. Potom si potiahne z ťahacej kopy päť kariet do ruky a tým ukončí svoj ťah.

Koniec hry nastane na konci ťahu hráča, ak už zásoba neobsahuje žiadnu kartu *Province*, alebo ak sa zo zásoby minuli tri druhy kariet.

Keď nastane situácia, že si má hráč potiahnuť kartu z prázdnej ťahacej kôpky, premieša sa jeho odhadzovacia kôpka a vytvorí sa z nej nová ťahacia. Použité karty sa takto opakovane vracajú do ruky hráča.

Okrem základných typov kariet existujú ešte karty reakcií (*reaction cards*), ktoré môže hráč zahrať mimo svojho ťahu ako reakciu na nejaký podnet, najčastejšie ako obranu pred útokom iného hráča.

1.2 Formálny zápis hry

Začneme pripomenutím základných pojmov teórie hier (budeme pri tom vychádzať z [9]). Potom formálne zapíšeme stav hry, povolené akcie hráčov, ich informačné množiny a zisky.

Naším cieľom nie je podrobne zachytiť všetky pravidlá Dominionu (to by na tomto priestore vzhľadom na ich komplexnosť ani nebolo možné), ale skôr poskytnúť akúsi kostru, na ktorej by sa (v prípade potreby a záujmu) dalo ďalej stavať.

1.2.1 Základné pojmy

Hráč je jednotlivec, ktorý vykonáva rozhodnutia. Možnosti, z ktorých si hráč vyberá, nazývame **akcie**. Keďže priebeh hry Dominion závisí aj od náhodných dejov (miešanie kariet na začiatku a po vyčerpaní ťahacej kopy), zaveďme pseudohráča **Náhoda**, ktorý bude tieto deje vykonávať ako svoje akcie, pričom sa bude riadiť danými pravdepodobnosťami.

Stav hry zachytáva všetky aspekty hry, ktoré boli ovplyvnené vykonanými akciami hráčov, teda sa v čase mení. Prechod k novému stavu je možný iba vybratím akcie určeným hráčom (tzv. hráčom na ťahu). **Koncový stav** je stav, z ktorého nie je možné vykonať žiadnu akciu.

Zisk hráča je definovaný pre každý koncový stav a každého hráča. Cieľom hráčov je maximalizovať svoj zisk.

Informáciu hráča budeme modelovať **informačnou množinou**, t.j. množinou všetkých stavov, v ktorých sa hra podľa informácií prístupných hráčovi môže nachádzať. **Stratégia** hráča určuje, ktorú akciu hráč zvolí, ak má k dispozícii danú informačnú množinu. Je to teda zobrazenie z množiny informačných množín do množiny akcií.

Keďže v Dominione hráči konajú opakovane v kolách, a teda sa môžu rozhodovať aj na základe už vykonaných akcií, na popis hry použijeme **extenzívnu formu**. Tá sa skladá z nasledujúcich častí:

- strom, ktorého vrcholom sú priradené stavy hry (v koreni je počiatočný stav, v listoch koncové stavy) a hranám akcie,
- priradenie hráčov vrcholom (hráč na ťahu),
- pravdepodobnosti, ktorými sa riadi Náhoda pri výbere akcií vo svojich vrcholoch,
- informačné množiny, do ktorých sú rozdelené vrcholy hráčov,
- zisky hráčov v koncových vrcholoch.

Informáciu hráčov môžeme na základe extenzívnej formy kategorizovať podľa štyroch kritérií:

- dokonalá informácia – každá informačná množina je jednoprvková,
- istota – všetky ťahy Náhody prebiehajú skôr než ktorýkoľvek ťah skutočných hráčov,
- symetrická informácia – informačná množina hráča v momente jeho ťahu a tiež v koncovom vrchole obsahuje aspoň tie prvky ako informačná množina ľubovoľného iného hráča (t.j. hráč nemá súkromnú informáciu),
- úplná informácia – Náhoda neťahá ako prvý hráč, alebo jej prvý ťah je známy každému hráčovi.

Podľa týchto kritérií je Dominion hrou bez istoty (Náhoda počas hry opakovane rozhoduje o poradí kariet v hráčovej ťahacej kôpke), s asymetrickou informáciou (hráč na ťahu ako jediný pozná svoje karty v ruke), s neúplnou informáciou (Náhoda na začiatku rozhodne, ktoré karty dostanú hráči do ruky), a teda s nedokonalou informáciou.

1.2.2 Karta

Hoci hráči nerozoznávajú karty s rovnakým menom, v skutočnosti sú to rôzne entity. Preto budeme pracovať s prvkami množiny kariet, ktorú označíme C. Začnime však s pomocnými predikátmi a funkciami definovanými len na množine názvov kariet označenej C'.

Definujme pre všetky $x \in C'$ predikáty $\mathit{TreasureCard}(x)$, $\mathit{VictoryCard}(x)$, $\mathit{CurseCard}(x)$ a $\mathit{ActionCard}(x)$ podľa zodpovedajúcich typov kariet. Takto umožníme priradiť jednému názvu karty viacero typov, čo sa naozaj využíva v rozšíreniach Dominionu.

Definujme funkcie Cost, $\mathit{TreasureValue}$ a $\mathit{VictoryValue}$, všetky z C' do $\mathbb{Z}$, podľa cien a zodpovedajúcich hodnôt kariet,[1] pričom:

$$\forall x \in C' (\neg \mathit{TreasureCard}(x) \Rightarrow \mathit{TreasureValue}(x) = 0)$$

$$\forall x \in C' \left(\neg (\mathit{VictoryCard}(x) \vee \mathit{CurseCard}(x)) \Rightarrow \mathit{VictoryValue}(x) = 0\right)$$

[1] Takýto model je príliš slabý; neumožňuje napríklad existenciu karty *Gardens*, ktorej hodnota závisí od počtu kariet v hráčovom balíčku. Pre jednoduchosť ho ale budeme ďalej používať.

Na priradenie názvu karte použijeme funkciu $Name$: $C \to C'$. Teraz môžeme vyššie definované predikáty a funkcie rozšíriť aj na jednotlivé karty, napríklad:

$$\forall x \in C(\mathit{TreasureCard}(x) \iff \mathit{TreasureCard}(\mathit{Name}(x)))$$

$$\forall x \in C(\mathit{Cost}(x) = \mathit{Cost}(\mathit{Name}(x)))$$

1.2.3 Stav neaktívneho hráča

Pod pojmom neaktívny hráč myslíme skutočného hráča (nie Náhodu), ktorý práve nie je na ťahu.

Jeho stav zapíšeme ako usporiadanú trojicu (D, A, H), kde $D = (c_1, \ldots, c_n)$ je konečná (možno prázdna) postupnosť kariet v ťahacej kope, A je množina kariet v odhadzovacej kope a H je množina kariet v ruke hráča.

Poradie kariet v D interpretujeme odspodu nahor: prvá karta v postupnosti je úplne naspodku ťahacej kopy, druhá karta je hneď nad ňou, ..., posledná karta v postupnosti zodpovedá vrchnej karte ťahacej kopy.

1.2.4 Stav hráča na ťahu

Stav hráča na ťahu zapíšeme ako usporiadanú sedmicu $(D, A, H, L, \alpha, \beta, \gamma)$, kde D, A a H majú rovnaký význam ako v stave neaktívneho hráča, L je množina zahraných kariet a $\alpha, \beta, \gamma \in \mathbb{N}$.

Hodnoty α, β, γ zodpovedajú postupne počtu akčných kariet, ktoré hráč ešte môže zahrať vo fáze akcií; počtu kariet, ktoré si ešte hráč môže kúpiť vo fáze nákupov; a celkovej peňažnej hodnote vyložených kariet peňazí a bonusov zo zahraných kariet akcií.

1.2.5 Stav hry

Stav hry Dominionu s n hráčmi ($n \in \{2, 3, 4\}$) a počiatočnou spoločnou zásobou S_0 je usporiadaná štvorica (S, d, P, f), kde S je množina kariet v spoločnej zásobe, $d \in \{1, \ldots, n\}$ je číslo hráča na ťahu, $P = (P_1, \ldots, P_n)$ je usporiadaná n-tica stavov hráčov a $f \in \{a, b_1, b_2, c\}$ je fáza, v ktorej sa nachádza hráč na ťahu.

Počiatočná spoločná zásoba S_0 obsahuje:

- $60 - 7n$ kariet *Copper*,
- 40 kariet *Silver*,
- 30 kariet *Gold*,
- 12 kariet *Estate* (pre $n = 2$ len 8 kariet),
- rovnako veľa kariet *Duchy* ako kariet *Estate*
- rovnako veľa kariet *Province* ako kariet *Estate*
- $10n - 10$ kariet *Curse*,
- po 10 kusov z 10 rôznych druhov kráľovských kariet (výnimkou sú kráľovské karty víťazných bodov; tých je rovnako veľa ako kariet *Estate*).

Stav P_i je stavom hráča na ťahu, ak $i = d$; inak je stavom neaktívneho hráča. Hráč vo svojom ťahu prestrieda postupne fázy a (fáza akcií), b_1 (hranie kariet peňazí), b_2 (nákup) a c (fáza upratovania).

Počiatočný stav hry je $(S_0, 1, (P_1, \ldots, P_n), a)$, pričom každý hráč má vo svojej odhadzovacej kope 7 kariet *Copper* a 3 karty *Estate* (žiadna karta sa nenachádza na viac ako jednom mieste, vrátane spoločnej zásoby), všetky ostatné kopy sú prázdne a pre hráča na ťahu platí $\alpha = \beta = \gamma = 0$.

Koncové stavy sú také stavy (S, d, P, f), pre ktoré platí $f = c$ a zároveň aspoň jedna z podmienok:

- v S sa nenachádza žiadna karta *Province*,
- z S sa úplne minuli (oproti S_0) aspoň tri rôzne druhy kariet.

1.2.6 Akcie hráčov[2]

Stav hry, ako sme ho definovali, neurčuje jednoznačný vrchol stromu extenzívnej formy – hra sa môže dostať do toho istého stavu rôznymi postupnosťami akcií hráčov. To nám neprekáža; užitočné by však bolo, keby z každého vrcholu stromu s tým istým stavom hry viedli rovnaké akcie hráča. V tom prípade by sme akcie z vrcholu mohli definovať len podľa stavu hry v tomto vrchole.

S týmto cieľom sme do stavu hry pridali komponent f, fázu ťahu hráča. Delenie na štyri fázy však stále nedokáže rozlíšiť všetky situácie, aké môžu počas trvania hráčovho ťahu nastať.[3]

Tento problém obídeme tak, že budeme akcie hráčov definovať naraz vo väčších logických celkoch.[4] Vnútri rôznych celkov sa tak môžu objaviť viaceré vrcholy s rovnakým stavom hry a rôznymi množinami vychádzajúcich akcií hráčov, na hraniciach celkov však už bude situácia jednoznačná.

Akcie hráčov budeme zapisovať ako zmeny stavu hry. Pre zjednodušenie budeme používať iba primitívne zmeny; v prípade potreby môžeme zložitú zmenu rozdeliť na niekoľko čiastkových a tie združiť do jedného celku. (V strome extenzívnej formy sa to prejaví ako reťaz vrcholov, v ktorých má hráč na výber z jedinej akcie.)

Miešanie kariet Náhodou zapíšeme ako akciu $\mathit{Shuffle}^p_\pi$, kde p je číslo hráča a π je výsledné poradie kariet v ťahacej kope. Po aplikovaní tejto zmeny na stav hry $G = (S, d, P, f)$ dostaneme nový stav $\mathit{Shuffle}^p_\pi(G) = (S, d, P', f)$, ktorý sa od G líši iba v stave hráča P_p (jeho odhadzovacia kopa A'_p je prázdna a ťahacia kopa D'_p obsahuje karty pôvodnej odhadzovacej kopy A_p v poradí π).

Z každého vrcholu stromu, v ktorom má Náhoda miešať karty, povedie práve $|A_p|!$ takýchto akcií $\mathit{Shuffle}^p_\pi$. Každú z možných permutácií π vyberie Náhoda s rovnakou pravdepodobnosťou $1/|A_p|!$.

[2] Slovo *akcia* sa nám môže asociovať s dvoma nesúvisiacimi pojmami: *akcia hráča* je jedna z možností, medzi ktorými sa hráč pri svojom ťahu rozhoduje; *karta akcií* je konkrétny prvok hry Dominion. Budeme sa preto vyhýbať nešpecifikovanému pojmu *akcia* všade tam, kde by jeho použitie vnieslo nejednoznačnosť.

[3] Počas ťahania kariet sa môže ťahacia kôpka minúť a k slovu sa nakrátko dostane Náhoda, aby zamiešala odhadzovaciu kopu. Potom sa však na ťah opäť vráti pôvodný hráč.

Iným príkladom je zahranie zložitejších kariet akcií, kedy pravidlá vyžadujú viacnásobné rozhodovanie hráča na ťahu alebo aj ostatných hráčov.

[4] Takýmto celkom bude napríklad ťahanie karty z ťahacieho balíčka alebo vykonanie pokynov na karte akcií.

Potiahnutie jednej karty z ťahacej kopy do ruky hráča bude reprezentovať akcia Draw^p. Nech $D_p = (c_1, \ldots, c_n)$ je ťahacia kopa hráča p, A_p jeho odhadzovacia kopa a H_p ruka. Ak je D_p neprázdna, teda $n \geq 1$, potom:

$$\mathit{Draw}^p((S, d, P, f)) = (S, d, P', f)$$

$$D'_p = (c_1, \ldots, c_{n-1}) \wedge H'_p = H_p \cup \{c_n\}$$

a ostatné komponenty stavov hráčov sú nezmenené (podobne budeme zmeny stavov hry zapisovať aj ďalej).

Ak $D_p = ()$ a $A_p \neq \emptyset$, potom akciu Draw^p zložíme z dvoch čiastkových: $\mathit{Shuffle}^p_\pi$ (pre všetky možné π) a Draw^p. Ak $D_p = ()$ a $A_p = \emptyset$, akcia Draw^p ponechá stav hry bez zmeny.

Zahranie karty z ruky zapíšeme akciou Play_c, kde c je hraná karta. Túto akciu môže vykonať jedine hráč na ťahu, preto na rozdiel od minulých prípadov nemusíme špecifikovať dotknutého hráča p. Nech H_d je ruka hráča na ťahu a L_d sú jeho zahrané karty, potom:

$$\mathit{Play}_c((S, d, P, f)) = (S, d, P', f)$$

$$H'_d = H_d \setminus \{c\} \wedge L'_d = L_d \cup \{c\}$$

Získanie karty c zo spoločnej zásoby do odhadzovacej kopy A_p hráča p bude predstavovať akcia Gain^p_c:

$$\mathit{Gain}^p_c((S, d, P, f)) = (S \setminus \{c\}, d, P', f)$$

$$A'_p = A_p \cup \{c\}$$

Na zmenu hodnôt α, β, γ hráča na ťahu budeme používať akcie $\mathit{Set}\text{-}\alpha_x$, $\mathit{Set}\text{-}\beta_x$, $\mathit{Set}\text{-}\gamma_x$, kde x je nová hodnota. Napríklad:

$$\mathit{Set}\text{-}\alpha_x((S, d, P, f)) = (S, d, P', f)$$

$$\alpha' = x$$

Kúpu karty zo spoločnej zásoby (Buy_c) dostaneme ako zloženie získania karty zo zásoby a zmien hodnôt β, γ hráča na ťahu: Gain^d_c, $\mathit{Set}\text{-}\beta_{\beta'}$, $\mathit{Set}\text{-}\gamma_{\gamma'}$, kde $\beta' = \beta - 1$ a $\gamma' = \gamma - \mathit{Cost}(c)$.

Podobne treba definovať akcie zodpovedné za presuny kariet medzi rôznymi kôpkami, upratovaciu fázu, zmenu hráča na ťahu, vykonanie jednotlivých kariet akcií, ...

1.2.7 Strom extenzívnej formy

V koreni stromu je počiatočný stav hry a na ťahu Náhoda, ktorá postupne premieša každému hráčovi jeho odhadzovaciu kôpku a vytvorí mu z nej ťahaciu kôpku. Následne si hráči potiahnu do ruky päť kariet, hráčom na ťahu sa stane hráč 1 a nastaví sa mu $\alpha = \beta = 1$.

Túto inicializáciu hry môžeme zapísať pomocou akcií hráčov definovaných vyššie: z koreňa budú vychádzať hrany $\mathit{Shuffle}^1_\pi$ pre všetky π; z ich koncových vrcholov povedú hrany $\mathit{Shuffle}^2_\pi$, atď. až po $\mathit{Shuffle}^n_\pi$, kde n je počet hráčov.

Ďalej povedú reťaze akcií $Draw^1, \ldots, Draw^n$; z každej päť kusov. Nakoniec ešte akciami $Set\text{-}\alpha_1$ a $Set\text{-}\beta_1$ nastavíme hodnoty α, β hráčovi na ťahu a tak ukončíme logický celok inicializácia.

Časti stromu, ktoré reprezentujú fázu akcií hry Dominionu, začínajú vrcholmi so stavom hry v tvare (S, d, P, a). Ak je v stave P_d hráča na ťahu hodnota $\alpha \geq 1$, z týchto vrcholov budú vychádzať hrany

$$\{Play_c \mid c \in H_d \wedge ActionCard(c)\}$$

nasledované $Set\text{-}\alpha_{-1}$ a reťazou hrán vykonávajúcich príslušnú akciu karty c. Navyše z takýchto vrcholov povedú hrany meniace fázu ťahu na b_1.

Ak je $\alpha = 0$, hráčovi zostane jediná možná akcia: presunúť sa do fázy b_1.

Hrany vychádzajúce zo stavu (S, d, P, b_1) budú pozostávať zo zmeny fázy ťahu na b_2 a z akcií

$$\{Play_c \mid c \in H_d \wedge TreasureCard(c)\}$$

nasledovaných akciami $Set\text{-}\gamma_{\gamma'}$, kde $\gamma' = \gamma + TreasureValue(c)$ pre príslušnú kartu c.

Zo stavu v tvare (S, d, P, b_2) povedú v prípade $\beta \geq 1$ akcie

$$\{Buy_c \mid c \in S \wedge Cost(c) \leq \gamma\}$$

a tiež zmena fázy ťahu na c; v prípade $\beta = 0$ iba zmena fázy ťahu.

Fáza upratovania nevyžaduje hráčovo rozhodovanie, preto zo stavov (S, d, P, c) budú vychádzať iba reťaze[5] akcií: presun všetkých kariet z ruky hráča a zahraných kariet do odhadzovacej kopy, ťahanie nových piatich kariet do ruky, zmena hráča na ťahu, inicializovanie fázy ťahu a hodnôt α, β, γ.

1.2.8 Informačné množiny

Keďže informačné množiny hráča p tvoria rozklad množiny vrcholov stromu extenzívnej formy, môžeme ich reprezentovať aj reláciou ekvivalencie $\equiv_p$ medzi vrcholmi.

V našom modeli hry predpokladáme, že hráči nezabúdajú históriu vykonaných akcií. Nutnou podmienkou na ekvivalenciu dvoch vrcholov (rôznych od koreňa) je preto ekvivalencia ich predkov v strome.

Navyše efekty akcií, ktorými sme sa do týchto vrcholov z ich predkov dostali, musia byť z pohľadu hráča p navzájom nerozoznateľné. Toto budeme značiť reláciou ekvivalencie $\sim_p$ medzi usporiadanými dvojicami (G, A), kde G je stav hry a A je akcia hráča.

To, ktorú permutáciu kariet si Náhoda pri miešaní odhadzovacej kopy hráča q vybrala, nepozná žiaden z hráčov, preto pre ľubovoľné permutácie π, ρ platí:

$$(G_1, Shuffle^q_\pi) \sim_p (G_2, Shuffle^q_\rho)$$

[5]Nie tak úplne doslova. Počas ťahania kariet do ruky sa môže „reťaz“ rozvetviť kvôli miešaniu kariet Náhodou pri vyčerpaní ťahacej kopy.

S ťahaním karty do ruky hráča q je situácia odlišná. Hráč q totiž túto kartu ako jediný po potiahnutí vidí (c_1, c_2 označujú vrchné karty ťahacích kôp hráča q v stavoch G_1, G_2):

$$(p \neq q \vee \mathit{Name}(c_1) = \mathit{Name}(c_2)) \iff (G_1, \mathit{Draw}^q) \sim_p (G_2, \mathit{Draw}^q)$$

Zahranie karty z ruky, získanie karty zo spoločnej zásoby a zmena hodnôt α, β, γ sú úplne verejné akcie:

$$\mathit{Name}(c_1) = \mathit{Name}(c_2) \iff (G_1, \mathit{Play}_{c_1}) \sim_p (G_2, \mathit{Play}_{c_2})$$

$$\mathit{Name}(c_1) = \mathit{Name}(c_2) \iff (G_1, \mathit{Gain}^q_{c_1}) \sim_p (G_2, \mathit{Gain}^q_{c_2})$$

$$(G_1, \mathit{Set}\text{-}\alpha_x) \sim_p (G_2, \mathit{Set}\text{-}\alpha_x)$$

Reláciu $\sim_p$ treba ďalej dodefinovať pre všetky akcie použité v strome.

1.2.9 Zisky v koncových vrcholoch stromu

Koncové vrcholy stromu sú vrcholy s koncovými stavmi hry. Zisky v nich reprezentujú výsledok hry – v prípade Dominionu sú možnými výsledkami len výhra a prehra jednotlivých hráčov, čo budeme značiť hodnotami 1 a 0.

Najprv si definujme pre každého hráča p počet jeho víťazných bodov w_p:

$$w_p = \sum_{c \in T_p} \mathit{VictoryValue}(c),$$

kde T_p je množina všetkých kariet hráča p (pre hráča na ťahu platí $T_p = D_p \cup A_p \cup H_p \cup L_p$, pre ostatných hráčov $T_p = D_p \cup A_p \cup H_p$).

Nech je v koncovom stave na ťahu hráč d. Potom zisk z_p hráča p vypočítame:

$$z_p = \begin{cases} 0 & \text{ak } \exists q \in \{1, \ldots, n\}(w_p < w_q \vee (w_p = w_q \wedge p \leq d < q)), \\ 1 & \text{inak} \end{cases}$$

Hru teda vyhrajú hráči s najväčším počtom víťazných bodov; v prípade rovnosti tí, ktorí odohrali menej ťahov.

Kapitola 2

Umelá inteligencia

V tejto kapitole zadefinujeme základné pojmy umelej inteligencie, rozdelíme inteligentných agentov a prostredia do kategórií a upresníme, kde sa v tomto rozdelení nachádza prostredie hry Dominion.

Celý tento prehľad je spracovaný podľa [10].

2.1 Základné pojmy

Agent je indivíduum, ktoré vníma svoje prostredie pomocou senzorov a ovplyvňuje ho pomocou efektorov. Napríklad ľudský agent má ako senzory oči, uši, hmatové orgány, ..., a ako efektory ruky, nohy, ústa, ...

Automatický hráč Dominionu nemá žiadne (ani virtuálne) orgány, ktoré by sa dali rozdeliť na senzory a efektory. Stále je však zmysluplné hovoriť priamo o jeho vnemoch a akciách. Hráč vníma napríklad názvy kariet, ktoré si potiahol do ruky, alebo kúpu karty zo spoločnej zásoby iným hráčom. Medzi hráčove akcie patrí napríklad zahranie útočnej karty.

Zmyslom agentovej činnosti býva splnenie určitého cieľa. Aby sme mohli posudzovať jeho úspešnosť, musíme si zaviesť nejakú mieru výkonu (*performance measure*) agenta.

Pre hráča Dominionu sa nám intuitívne núka veľmi jednoduchá binárna miera: Hráč je úspešný práve vtedy, keď vyhral hru. Pre niektoré účely je však táto miera nevhodná, lebo nerozlišuje mizerných agentov veselo kupujúcich karty *Curse* od pomerne inteligentných agentov, ktorí vždy končia hru v tesnom závese za víťazom. Preto budeme neskôr používať aj inú mieru, ktorá bude odzrkadľovať počet získaných víťazných bodov.

2.2 Typy agentov

Teoreticky by bolo možné vybudovať ideálneho agenta, ktorý by konal podľa vopred pripravenej tabuľky s pokynmi. Tabuľka by musela obsahovať pre každú prípustnú postupnosť vnemov riadok s ideálnou akciu, ktorú má agent vykonať. Agentovi by stačilo pamätať si históriu vnemov a zakaždým vyhľadať správnu akciu v tabuľke.

Tento prístup však trpí zjavným neduhom: Potrebná tabuľka býva príliš veľká – je prakticky nemožné vytvoriť ju a vtesnať do pamäte agenta. Uvedieme preto iné možnosti ako implementovať mapovanie vnemov na akcie.

Jednoduchý reflexívny agent vyberá svoje akcie len na základe aktuálne prítomných vnemov. Nepamätá si teda históriu vnemov ani stav prostredia, čo značne zjednodušuje jeho implementáciu. Tvorcovi agenta tiež stačí pripraviť kratší zoznam pravidiel.

Ak aktuálne vnemy neposkytujú dostatok informácií o prostredí, môžeme použiť **reflexívneho agenta s vnútorným stavom**. Ten si pamätá popis stavu prostredia a upravuje ho podľa prichádzajúcich vnemov. Pri rozhodovaní potom využíva zapamätaný stav.

Oproti agentom s vopred pripravenými pokynmi sú oveľa flexibilnejší **agenti snažiaci sa dosiahnuť svoj cieľ** (*goal-based agents*). Svoje akcie si plánujú v dlhších postupnostiach, prehľadávajú strom možného vývoja prostredia a riadia sa snahou splniť cieľ.

V tejto práci vytvoríme iba reflexívnych agentov, väčšina z nich bude dokonca bez vnútorného stavu. Agenti s cieľom na zreteli by síce mali väčšiu šancu pri hrách s ľudským protivníkom, no ich implementácia by bola o poznanie náročnejšia.

2.3 Typy prostredí

Prostredie, v ktorom agent koná, môžeme charakterizovať podľa viacerých kritérií.

V **dosiahnuteľnom** prostredí sú všetky podstatné aspekty sveta prístupné agentovi prostredníctvom jeho senzorov. Takéto prostredie je veľmi pohodlné, lebo agent si nepotrebuje pamätať stav.

Ak vývoj prostredia závisí iba od aktuálneho stavu a agentových akcií a dá sa na ich základe predpovedať, prostredie je **deterministické**. Toto kritérium sa často posudzuje z hľadiska agenta; v tom prípade sú spravidla komplexné nedosiahnuteľné prostredia aj nedeterministické.

Agentovo pôsobenie sa v **epizodickom** prostredí delí na nezávislé epizódy. Jeho rozhodnutia ovplyvňujú stav sveta iba v aktuálnej epizóde. Agent teda nemusí myslieť do budúcnosti.

Ak prostredie nečaká na agentovo rozhodnutie, ale sa mení, nazývame ho **dynamické**. V opačnom prípade je **statické** a agent pri rozhodovaní nie je časovo obmedzený.

Ak je množina vnemov a akcií obmedzená (konečná), prostredie nazývame **diskrétne**, inak je **spojité**.

Prostredie hry Dominion je pre hráča nedosiahnuteľné, nedeterministické, neepizodické, statické a diskrétne.

Kapitola 3

Evolučné algoritmy

Evolučné algoritmy sú metódy riešenia zväčša ťažkých optimalizačných problémov, ktoré sú inšpirované dejom biologickej evolúcie v prírode ([3]).

Počas raného vývoja tohto odvetvia informatiky vzniklo nezávisle viacero techník, k najznámejším patria evolučné stratégie, evolučné programovanie a genetické algoritmy. Hoci sa od seba v mnohých bodoch líšia, v princípe sú založené na rovnakej myšlienke. Ich porovnanie možno nájsť v [1].

V tejto kapitole predstavíme spoločný princíp menovaných techník spracovaný podľa [1, 2].

3.1 Kostra evolučného algoritmu

Označme priestor prípustných riešení daného optimalizačného problému ako Ξ a cieľovú funkciu (*objective function*) ako $f\colon \Xi \to \mathbb{R}$. Naším cieľom je nájsť riešenie $\xi \in \Xi$, ktoré maximalizuje hodnotu $f(\xi)$.

Budeme si udržiavať množinu riešení – populáciu, jej veľkosť označíme μ. V cykloch zvaných generácie vytvoríme z aktuálnej populácie λ nových jedincov a vyberieme μ z nich do novej populácie. Tento druh výberu sa označuje (μ, λ). Ak vyberáme aj z jedincov rodičovskej generácie, označujeme to $(\mu + \lambda)$.

Do počiatočnej populácie $P(0)$ sa často vyberajú náhodné prvky Ξ. Nové jedince vznikajú použitím genetických operátorov: Rekombinácia vytvorí nové riešenie skombinovaním viacerých rodičov a mutácia (zväčša náhodne) pozmení jedinca.

Na hodnotenie kvality jedinca sa používa fitnes funkcia $\Phi\colon \Xi \to \mathbb{R}$. Tá zohľadňuje cieľovú funkciu f, no môže riešenia hodnotiť aj podľa ďalších kritérií. V článku [1] nie sú ako prvky populácie použité priamo riešenia problému, ale všeobecnejšie indivíduá obsahujúce riešenie ako jednu zo svojich zložiek. V takom prípade je rozlišovanie fitnes a cieľovej funkcie nutné.

Nové generácie sa vytvárajú dovtedy, kým nie je splnená terminálna podmienka. Najčastejšie ňou býva dosiahnutie určitého počtu generácií alebo dostatočnej kvality riešenia.

Kostra evolučného algoritmu teda vyzerá nasledovne:

$t \leftarrow 0$
inicializácia $P(0)$
vyhodnotenie fitnes funkcie na $P(0)$
kým nie je splnená terminálna podmienka:
    $P'(t) \leftarrow$ rekombinácia $P(t)$
    $P''(t) \leftarrow$ mutácia $P'(t)$
    vyhodnotenie fitnes funkcie na $P''(t)$
    výber novej populácie $P(t+1)$
    $t \leftarrow t+1$

Kapitola 4

Implementácia herného prostredia

Na testovanie a porovnávanie rôznych stratégií je potrebná implementácia pravidiel Dominionu vo forme herného prostredia. V nasledujúcej kapitole popíšeme použité technológie a objektový návrh našej implementácie.

4.1 Použité technológie

Herné prostredie sme implementovali v programovacom jazyku C++ (verzia štandardu z roku 1998). Dôvodmi tohto výberu boli rozšírenosť jazyka, existencia širokej palety knižníc, no najmä efektivita výsledného binárneho kódu.

C++ je multiparadigmový jazyk. My sme si vybrali objektovo orientované programovanie, čo je pri takomto rozsiahlom projekte viac-menej nutnosťou.

Okrem štandardnej knižnice a STL používame v kóde niektoré z knižníc Boost. Na správu dynamicky alokovaných objektov slúžia inteligentné pointre `scoped_ptr` a `shared_ptr`, ktoré sa starajú o uvoľnenie pamäte v čase zániku spravovaného objektu.

Trieda `scoped_ptr` je pri práci so zapuzdreným pointrom efektívnejšia, no neumožňuje kopírovanie, a teda sa ani nedá používať v kontajneroch STL. V takých prípadoch preto používame `shared_ptr`, inak je `scoped_ptr` preferovanou možnosťou.

Ako generátor náhodných čísel používame Mersenne Twister z knižnice Boost.Random, konkrétne `mt19937`. Jeho výhodou je veľká perióda (až $2^{19937} - 1$) a rýchlosť, s akou generuje náhodné čísla. Knižnica Boost.Random navyše obsahuje implementáciu viacerých rozdelení náhodných premenných, z nich využijeme rovnomerné a normálne rozdelenie.

Na serializáciu stratégií, ktoré sú výsledkom evolučných procesov, používame knižnicu Boost.Serialization. Vďaka nej môžeme získané stratégie uložiť na disk, znova načítať a následne aj porovnávať navzájom.

4.2 Triedy objektového návrhu

V tejto časti predstavíme základné stavebné bloky našej implementácie, ich rozhrania a prepojenia.

4.2.1 Trieda `Card`

Pretože karty s rovnakým názvom sú od seba neodlíšiteľné, použili sme návrhový vzor jedináčik (*singleton*). Každý druh karty zastupuje jediná inštancia triedy pomenovanej rovnako ako karta. Rôzne výskyty tohto druhu karty potom reprezentujeme pointermi na jej inštanciu.

Abstraktná trieda `Card` definuje rozhranie, ktoré musí spĺňať každá karta:

`static Card* Get()`
: vráti pointer na jedinú inštanciu karty. Jedinečnosť je zaručená tým, že trieda `Card` a jej dcéry majú súkromné konštruktory.

`string name()`
: vráti názov karty.

`int cost()`
: vráti cenu karty.

`bool is_treasure_card(), bool is_victory_card(), ...`
: popisujú, do ktorých kategórií karta patrí.

`int victory_value(PlayerState const&)`
: vráti hodnotu karty vo víťazných bodoch. Ako argument berie stav hráča, pretože táto hodnota môže závisieť aj od ostatných kariet v kôpke (napríklad v prípade *Gardens*).

`int treasure_value(PlayerState const&)`
: vráti peňažnú hodnotu karty.

`void PerformAction(Arbiter*)`
: vykoná akciu podľa pokynov na karte. Keďže ako argument dostane arbitra, môže meniť stav ľubovoľného hráča.

`void PerformReaction(Arbiter*)`
: vykoná reakciu podľa pokynov na karte.

Napríklad cenu karty *Gold* získame výrazom `Gold::Get()->cost()`.

4.2.2 Trieda `Pile`

Na reprezentovanie neusporiadanej množiny kariet používame inštancie triedy `Pile`. Trieda poskytuje okrem základných metód na pridávanie a odoberanie kariet (`void Add(Card*)`, `void Add(Pile const&)`, `void Remove(Card*)`, `void Remove(Pile const&)`, ...) aj iterátory a rôzne pomocné metódy, napríklad:

`Card* GetRandomCard(RandomGenerator*)`
: vráti náhodnú kartu z kôpky.

`Pile FilterActionCards()`
: vráti tie karty kôpky, ktoré sú kartami akcií.

`Pile FilterCostInRangeCards(int, int)`
vráti tie karty kôpky, ktorých cena patrí do daného rozsahu.

`Pile GetIntersection(Pile const&)`
vráti prienik tejto kôpky s kôpkou v argumente.

4.2.3 Trieda `PlayerState`

Aktuálny stav hráčov si hráči pamätajú v inštanciách triedy `PlayerState`. Tie obsahujú jednotlivé kôpky (`deck`, `discard`, `hand`, `played`, `aside` a tiež `total` so všetkými kartami hráča), počet akcií, nákupov a peňazí (`actions`, `buys`, `coins`) a stav imunity proti útokom (`immune_to_attack`). K ťahacej kôpke je navyše priradené aj poradie kariet.

Trieda umožňuje presúvať karty medzi kôpkami sadou metód `Move`, pričom ako zdrojová kôpka môže vystupovať aj spoločná zásoba a ako cieľová kôpka smetisko.

V prípade, že názov presúvanej karty nepoznáme, použijeme špeciálny druh karty `UnknownCard`. Stav iného hráča takto nemusí byť úplný, ale popisuje len tú informáciu, ktorú o ňom máme. Na prácu s kartami `UnknownCard` je prispôsobená aj trieda `Pile`.

Okrem toho trieda obsahuje metódy:

`int CountVictoryPoints()`
vráti celkový počet víťazných bodov hráča.

`void CleanUp()`
vykoná fázu čistenia.

`void ShuffleDiscardIntoDeck()`
zamieša odhadzovaciu kopu a vytvorí z nej ťahaciu. Pretože trieda `PlayerState` slúži pre samotných hráčov, ktorí nemajú žiadnu informáciu o výsledku miešania, táto metóda v skutočnosti nevytvorí reálne poradie kariet v ťahacej kôpke, len si poznačí, že ho nepozná.[1]

Od `PlayerState` je odvodená trieda `CompletePlayerState`, ktorá je určená na pamätanie si stavu hráčov arbiterom. V nej je táto metóda predefinovaná tak, aby naozaj zamiešala karty odhadzovacej kôpky a vytvorila nové poradie kariet v ťahacej kôpke.

4.2.4 Trieda `Player`

Trieda `Player` obsahuje informácie o stave hry, ktoré sú prístupné hráčovi a môže sa podľa nich riadiť stratégia. Pamätá si počet hráčov hry, obsah spoločnej zásoby kariet, stav hráča a súperov.

Trieda ďalej stratégii poskytuje pomocné metódy:

`Pile GetActionsInHand()`
vráti karty akcií, ktoré má hráč v ruke.

[1] Časom ale hráč môže zistiť aspoň nejaké informácie o svojej alebo súperovej ťahacej kôpke. Napríklad zahraním karty *Bureaucrat* dostane hráč na ťahu na vrch svojej ťahacej kôpky kartu *Silver*, čo je verejná akcia. Neskôr, keď si tento hráč potiahne do ruky karty, všetci vedia, že aspoň jedna z kariet v jeho ruke je *Silver*.

`Pile GetBuyableCards()`
: vráti karty spoločnej zásoby, ktoré si hráč môže dovoliť kúpiť.

`int EstimateGameEnd()`
: odhadne podľa súčasného obsahu spoločnej zásoby, ako ďaleko je koniec hry. Tento odhad je založený na minimálnom počte a cene kariet, ktoré treba kúpiť, aby boli splnené podmienky ukončenia hry.

  V priebehu hry vrátené hodnoty postupne klesajú. Ak je splnená podmienka na ukončenie hry, odhad je rovný 0.

4.2.5 Trieda `Query`

Počas hrania niektorých kariet akcií sa dáva ťahajúcemu hráčovi (niekedy aj ostatným) na výber z viacerých možností. Tento proces zachytávajú inštancie triedy `Query` a jej dcér.

Priamymi potomkami sú triedy `YesNoQuestion` (pre otázky s možnými odpoveďami áno a nie), `SingleCardChoice` (pre výber jednej karty zo zadaných možností) a `MultipleCardsChoice` (pre výber viacerých kariet zo zadaných možností; špecifikovaný je aj minimálny a maximálny počet kariet vo výbere).

Príklad použitia: Karta *Chapel* dáva ako svoju akciu hráčovi možnosť vyhodiť najviac štyri karty z ruky na smetisko. Kód vykonávajúci túto akciu preto vytvorí inštanciu triedy `TrashCardsFromHandQ`, ktorá je odvodená od `MultipleCardsChoice`, pričom jej dá ako parametre množinu kariet v ruke, minimálny počet vybraných kariet 0 a maximálny počet 4. Tento objekt potom posunie stratégii (konkrétne metóde `Respond`), ktorá do neho uloží svoj výber.

Každý objekt typu `Query` má hneď po svojej inicializácii prednastavenú nejakú validnú odpoveď. Stratégia tak nemusí vedieť odpovedať na všetky otázky, aké môže dostať.

4.2.6 Trieda `Strategy`

Táto trieda reprezentuje samotnú logiku hráča. Využíva objekt `Player` na získavanie informácií o stave hry a vykonáva všetky rozhodnutia.

Trieda `Strategy` je abstraktná a definuje rozhranie, ktoré musí spĺňať každá implementácia stratégie:

`Card* PlayActionCard()`
: vráti kartu akcií v ruke hráča, ktorú chce zahrať vo svojej fáze akcií; alebo špeciálnu kartu `NoCard::Get()`, ak nechce zahrať žiadnu akciu.

`Card* PlayTreasureCard()`
: vráti kartu peňazí v ruke hráča, ktorú chce zahrať počas vykladania kariet vo fáze nákupov; alebo `NoCard::Get()`. Táto metóda je už v triede `Strategy` implementovaná tak, že vráti prvú kartu peňazí v ruke, čo by malo väčšine stratégií vyhovovať.

`Card* BuyCard()`
: vráti kartu v spoločnej zásobe, ktorú si chce hráč kúpiť počas svojej fázy nákupov; alebo `NoCard::Get()`.

`void Respond(Query*)`
: a ďalšie verzie pre každú z tried odvodených od `Query` odpovedajú na danú otázku.

4.2.7 Trieda `Arbiter`

Trieda `Arbiter` predstavuje rozhodcu – pamätá si presný aktuálny stav každého hráča (`CompletePlayerState`), v kolách vyzýva hráčov k ťahu a kontroluje dodržiavanie pravidiel hry.

Verejné metódy, ktorými sa dá meniť stav jednotlivých hráčov (napríklad presúvať karty medzi kôpkami), slúžia pre kód vykonávajúci akcie kariet. Tým je umožnené jednoducho pridávať karty z ďalších rozšírení Dominionu bez toho, aby sa upravoval kód arbitra.

Použitie inštancie triedy `Arbiter` jej vlastníkom sa limituje na predanie zoznamu stratégií, počiatočnej spoločnej zásoby kariet a generátora náhodných čísel konštruktoru a volanie metódy `PlayGame`, ktorá odsimuluje priebeh celej hry. Metóda `GetResults` potom vráti výsledky hry (počet víťazných bodov jednotlivých hráčov a zoznam víťazov).

Momentálna implementácia triedy `Arbiter` predstavuje tzv. lokálneho arbitra – vyžaduje absolútnu znalosť stavu hry. Bolo by možné vytvoriť aj vzdialeného arbitra, ktorý by sa pripojil na server s cudzou implementáciou hry Dominion[2] a sprostredkúval by komunikáciu medzi lokálnymi hráčmi a serverom.

[2] Napríklad http://dominion.isotropic.org/.

Kapitola 5

Stratégie

Povedané terminológiou našej implementácie herného prostredia Dominionu, cieľom tejto práce je vyvinúť čo najlepšiu realizáciu abstraktnej triedy `Strategy`.

Keďže štýl hry a teda aj optimálna stratégia značne závisia od toho, ktorých desať kráľovských kariet si hráči pred hrou zvolili, nesnažíme sa vytvoriť univerzálnu stratégiu. Namiesto toho postavíme model stratégie a pomocou evolučných algoritmov necháme vyvinúť jeho parametre priamo na mieru konkrétnej desatice kariet.

V tejto kapitole popíšeme modely stratégií, ktoré sme vytvorili, ich parametre a implementáciu evolučných procesov.

5.1 Modely s parametrami

Pretože parametre našich modelov vyvinie evolúcia, na ich pomenovanie budeme používať terminológiu požičanú z evolučných algoritmov.

Gén je najmenšia jednotka, na ktorú má zmysel parametre rozdeliť. V našom prípade bude napríklad génom kladné reálne číslo. Množina logicky súvisiacich génov tvorí **chromozóm**. Množinu všetkých chromozómov budeme nazývať **genóm**.

Abstraktná trieda `Chromosome` určuje rozhranie, ktoré musí implementovať každý chromozóm:

`string ConvertToString()`
: vráti textovú reprezentáciu génov chromozómu v tvare čitateľnom pre človeka.

`shared_ptr<Chromosome> Mutate(double, double, RandomGenerator*)`
: vráti nový chromozóm, ktorý je mutáciou pôvodného. Prvým argumentom je tempo (*mutation rate*) a druhým sila mutácie (*mutation strength*).

  Tempo vyjadruje pravdepodobnosť mutácie jednotlivých génov. Podľa sily sa určuje veľkosť zmeny mutovaného génu. Zvyčajne to býva štandardná odchýlka normálneho rozdelenia zmeny, čo ale pri niektorých génoch nemusí dávať zmysel. Preto nie je parameter sily normalizovaný a rôzne chromozómy ho môžu interpretovať rôznymi spôsobmi.

Trieda `Genome` slúži ako kontajner chromozómov. Každý chromozóm musí mať priradený jedinečný identifikátor (názov). Trieda umožňuje pridávať, odoberať a iterovať chromozómy a volať metódu `Mutate` na všetkých chromozómoch, pričom vráti nový genóm – mutáciu pôvodného.

Rozhranie, ktoré musí spĺňať každý model stratégie s parametrami, špecifikuje abstraktná trieda `EvolvableStrategy`:

`shared_ptr<EvolvableStrategy> Create()`
: vráti novú inštanciu rovnakého modelu stratégie. Vďaka tejto metóde môžeme pri simulovaní evolúcie vytvoriť viacero kópií tej istej stratégie (a potom im zadať rôzne parametre).

`shared_ptr<Genome> GetInitialGenome(Pile const&, RandomGenerator*)`
: vráti genóm s parametrami pre túto stratégiu na základe počiatočného obsahu spoločnej zásoby kariet (prvý argument).

  Na začiatku evolúcie sa táto metóda μ-krát zavolá a vrátené genómy sa použijú na inicializáciu populácie.

`void set_genome(Genome*)`
: nastaví parametre stratégie.

5.1.1 Chromozómy

Trieda `RatiosChromosome`

Tento chromozóm predstavuje vzájomné pomery medzi prvkami, ktoré sú zakódované do reťazcov. V prípade kariet sa ako kódovanie používa ich názov.

Konštruktoru objektu `RatiosChromosome` zadáme množinu všetkých prípustných prvkov a záporné reálne číslo ℓ. Objekt si pre niektoré prvky pamätá priradené reálne číslo z intervalu $(0, 1]$, ostatné prvky má uložené v osobitnej množine ako nepoužité. Okrem degenerovaného prípadu, keď sú všetky prvky nepoužité, platí, že súčet priradených hodnôt je rovný 1. Hneď po inicializácii objektu sú všetky prvky nepoužité.

Existujú dva spôsoby mutácie tohto chromozómu, každá z nich sa udeje s polovičnou pravdepodobnosťou: Upravíme len pomery použitých prvkov, alebo zmeníme ich počet.

Úprava pomerov prebieha tak, že postupne každú z priradených hodnôt s pravdepodobnosťou rovnou tempu mutácie zmeníme – vynásobíme ju hodnotou e^x, kde x je náhodné číslo z normálneho rozdelenia so strednou hodnotou 0 a štandardnou odchýlkou rovnou sile mutácie.

Počet použitých prvkov môžeme zmeniť oboma smermi – pridať alebo odobrať prvok. V okrajových prípadoch (použili sme všetky, prípadne žiadne prvky) máme na výber len z jednej možnosti, no inokedy sa rozhodneme náhodne: Označme m počet použitých prvkov. S pravdepodobnosťou $e^{\ell m}$ pridáme nový prvok (náhodný z nepoužitých), inak náhodný prvok presunieme k nepoužitým. Novému prvku priradíme náhodné číslo z intervalu $(0, 1]$.

Nakoniec po vykonaní mutácie hodnoty v získanom chromozóme vydelíme ich súčtom a tak ich znormalizujeme (okrem degenerovaného prípadu).

Za hodnotu ℓ volíme väčšinou $-\frac{\ln 2}{k}$ pre nejaké $k \in \mathbb{Z}^+$. Potom ak má množina použitých prvkov veľkosť k, nový prvok sa pridá s polovičnou pravdepodobnosťou.

Trieda `RanksChromosome`

Tento chromozóm predstavuje poradie prvkov zakódovaných do reťazcov.

Rovnako ako to bolo v prípade `RatiosChromosome`, konštruktoru objektu `RanksChromosome` zadáme množinu všetkých prípustných prvkov a parameter ℓ. Niektoré prvky si bude tento objekt pamätať v nejakom poradí, zvyšné budú neusporiadané v množine nepoužitých prvkov.

Mutácia môže tiež prebiehať dvoma spôsobmi. Pridávanie a odoberanie prvkov sa riadi rovnakými pravdepodobnosťami ako v prípade `RatiosChromosome`. Pridaný prvok umiestnime na náhodnú pozíciu v poradí.

Zmena poradia vyzerá nasledovne. Každému prvku priradíme reálne číslo: s pravdepodobnosťou rovnou tempu mutácie to bude jeho pozícia v aktuálnom poradí sčítaná s $3x$, kde x je náhodné číslo z normálneho rozdelenia so strednou hodnotou 0 a štandardnou odchýlkou rovnou sile mutácie. V opačnom prípade bude prvku priradená hodnota rovná jeho pozícii v aktuálnom poradí.

Potom prvky zoradíme podľa priradených hodnôt (neklesajúco) a získame tak nové poradie.

Konštantu 3 sme zvolili empiricky tak, aby sa chromozómy `RatiosChromosome` a `RanksChromosome` pri rovnakej sile mutácie menili porovnateľne.

5.1.2 Šablóna modelov `ExtremalStrategy`

Všetky naše modely budú založené na rovnakom princípe: Stratégia odpovedá na podnety výberom najlepšej alebo najhoršej karty z množiny možností. To, ktorá karta je najlepšia, však závisí od kontextu – chceme ju získať, alebo zahrať?

Abstraktná trieda `Chooser` predstavuje nejaké (aj nedeterministické) hodnotenie kariet. Jej rozhranie vyzerá nasledovne:

`Card* GetBestCard(Pile const&, bool)`
: vráti najlepšiu kartu z daných možností (prvý argument). Ak je druhý argument `false`, metóda môže vrátiť aj `NoCard::Get()`. Podobne,

`Card* GetWorstCard(Pile const&, bool)`
: vráti najhoršiu kartu z daných možností.

`shared_ptr<Chooser> Create()`
: vytvorí novú inštanciu rovnakej dcérskej triedy.

`void GetInitialChromosomes(Genome*, Pile const&, RandomGenerator*)`
: vloží do genómu chromozómy s parametrami na základe počiatočného obsahu spoločnej zásoby kariet (druhý argument).

`void set_chromosomes(Genome*)`
: nastaví parametre hodnotenia.

Rôzne implementácie rozhrania triedy `Chooser` sú uvedené v nasledujúcich sekciách.

Inštancia triedy `ExtremalStrategy` dostane v konštruktore pointre na dve hodnotenia kariet: `gaining` a `playing`. Podľa nich potom reaguje na podnety od arbitra.

Hodnotenie `gaining` sa používa v kontexte vlastnenia kariet. Stratégia si podľa tohto hodnotenia napríklad kúpi najlepšiu kartu a na smetisko zahodí tú najhoršiu. Ak má zahodiť kartu súperovi, vyberie si tú najlepšiu.

V niektorých situáciách je výhodnejšie nekúpiť si (alebo nevyhodiť) žiadnu kartu. Ak má stratégia na výber aj túto možnosť, nastaví druhý argument metód `GetBestCard`/`GetWorstCard` na `false`.

Hodnotenie `playing` sa používa v kontexte hrania kariet. Stratégia podľa tohto hodnotenia napríklad hrá najlepšiu akciu a vyhadzuje po útoku kartou *Militia* z ruky do odhadzovacej kôpky najhoršie karty.

5.1.3 Hodnotenia kariet

Trieda `RandomChooser`

Najjednoduchším hodnotením kariet je `RandomChooser`. Metódy `GetBestCard` aj `GetWorstCard` má implementované rovnakým spôsobom: vracia náhodnú kartu z daných možností. Nemá žiadne parametre, preto do genómu nepridáva chromozómy.

Toto hodnotenie sa teda nedokáže vyvíjať, no má zmysel používať ho v kombinácii s iným hodnotením (napríklad ak chceme nechať výber hraných akcií na náhodu).

Trieda `WeightedRandomChooser`

Hodnotenie `WeightedRandomChooser` tiež vyberá karty náhodne, no pravdepodobnosti, s akými to robí, sú určené jeho parametrami. Do genómu pridáva jeden chromozóm typu `RatiosChromosome`.

Tento chromozóm určuje pomery pravdepodobností výberu jednotlivých kariet metódou `GetBestCard`. Hodnoty uložené v chromozóme sa normalizujú tak, aby ich súčet pre dané možnosti bol rovný 1. Napríklad ak máme na výber z kariet *Copper* s hodnotou 0.1 a *Silver* s hodnotou 0.3, tak s pravdepodobnosťou 0.25 vyberieme kartu *Copper* a s pravdepodobnosťou 0.75 kartu *Silver*.

V prípade, že je súčet hodnôt možností rovný 0, vrátime `NoCard::Get()`. Ak to nie je povolené, vrátime náhodnú z možností (každú s rovnakou pravdepodobnosťou).

Metóda `GetWorstCard` funguje podobne. Ak sú medzi možnosťami také, ktoré majú v chromozóme priradenú hodnotu 0, vrátime náhodnú z nich (každú s rovnakou pravdepodobnosťou); inak `NoCard::Get()`. Ak to nie je povolené, použijeme prevrátené hodnoty z chromozómu ako pomery pravdepodobností.

Trieda `RanksChooser`

Toto hodnotenie kariet používa na reprezentáciu svojich parametrov chromozóm `RanksChromosome`.

Metóda `GetBestCard` postupne prechádza poradie kariet v chromozóme a vráti prvú takú, ktorá sa nachádza medzi zadanými možnosťami. V prípade, že tam taká karta nie je, vráti `NoCard::Get()`. Ak to nie je povolené, vráti náhodnú z možností (každú s rovnakou pravdepodobnosťou).

Metóda `GetWorstCard` prioritne vráti náhodnú z kariet, ktoré sa nenachádzajú v zozname chromozómu. Ak také karty na výber nie sú, vráti `NoCard::Get()`.

Ak to má zakázané, vráti tú kartu z možností, ktorá je v poradí chromozómu čo najviac na konci.

Trieda `RatiosChooser`

Hodnotenie `RatiosChooser` si ukladá svoje parametre do chromozómu typu `RatiosChromosome` a interpretuje ich ako pomery počtov kariet jednotlivých druhov v ideálnom balíčku hráča.

Označme počet všetkých kariet hráča ako m, počet jeho kariet typu c ako a_c a hodnotu, ktorá je priradená karte c v chromozóme, ako r_c. Dopyt po karte c potom vypočítame výrazom $r_c m - a_c$.

Metóda `GetBestCard` vráti tú z možností, po ktorej je najväčší dopyt. Ak je to povolené, k možnostiam sa pridá aj `NoCard::Get()` s dopytom 0.

Metóda `GetWorstCard` funguje podobne, len vracia kartu, po ktorej je najmenší dopyt.

Trieda `SeasonalChooser`

Táto trieda je len kontajnerom pre iné typy hodnotenia. Umožňuje používať rôzne spôsoby výberu kariet podľa toho, v akej fáze sa hra nachádza.

V konštruktore dostane `SeasonalChooser` vektor pointrov na objekty typu `Chooser` a vektor celých čísel – zodpovedajúcich prahov. Metódy `GetBestCard` a `GetWorstCard` len posúvajú požiadavky aktuálnemu z uskladnených hodnotení.

To, ktoré hodnotenie je aktuálne, sa určuje podľa odhadu konca hry vypočítaného metódou `EstimateGameEnd()` objektu `Player`. Použijeme prvý taký `Chooser`, ktorého zodpovedajúci prah nie je väčší ako odhad konca hry.

Postupnosť prahov by teda mala byť klesajúca. Prah posledného hodnotenia je implicitne nastavený na 0.

5.2 Referenčné stratégie

Aby sme vedeli posúdiť kvalitu stratégií, ku ktorým sa dopracujeme, potrebujeme referenčné stratégie. Dve z nich okrem toho použijeme na výpočet fitnes funkcie pri simulovaní evolúcie.

5.2.1 Trieda `GreedyStrategy`

Správanie ľudského hráča, ktorý je v Dominione ešte len začiatočníkom, sa snaží napodobňovať stratégia `GreedyStrategy`.

Táto stratégia sa rozhoduje skoro výhradne podľa cien kariet. Pri nákupe si vyberie najdrahšiu kartu, na ktorú má dostatok peňazí, pričom si nikdy nekúpi karty *Curse* a *Copper*.[1] Ak má na výber viacero kariet s rovnakou cenou, zvolí si náhodnú. Podobne sa rozhoduje aj pri hraní kariet akcií.

Pri vyhadzovaní kariet na smetisko stratégia uprednostňuje karty *Curse*. Ak je nútená vyhodiť viac kariet, postupuje od najlacnejších. Najlacnejšou kartou odpovedá aj na výzvy po zahraní kariet *Remodel* a *Mine*.

[1] Tieto dve karty majú cenu 0, preto by si ich hráč mohol kúpiť, kedykoľvek by mal voľný nákup. Zvyčajne je ale výhodnejšie v takomto prípade nekúpiť nič. (Začiatočník by však mohol namietať a *Copper* by nikdy neodmietol.)

Ak má stratégia vyhodiť karty z ruky do odhadzovacej kôpky, začína od kariet víťazných bodov a potom ďalej postupuje od najlacnejších kariet.

Stratégia `GreedyStrategy` si teda kúpi kartu *Province*, ak má aspoň 8 peňazí; vo väčšine hier si takisto kúpi kartu *Gold*, ak nasporí aspoň 6 peňazí.

Okrem toho je však jej balík kariet príliš rôznorodý a zbytočne pestrý. Táto stratégia trpí nedostatkom kariet peňazí a prebytkom kariet akcií, z ktorých väčšinu počas svojho ťahu ani nedostane možnosť zahrať.

5.2.2 Trieda `BigMoney`

Túto stratégiu sme prebrali zo stránky [6]. Hraje na úrovni stredne pokročilého ľudského hráča a vo väčšine hier poráža začiatočníkov.

Stratégia `BigMoney` ignoruje karty akcií a sústreďuje sa len na kupovanie kariet peňazí a víťazných bodov. Vďaka tomu sa dá použiť v každej hre, na druhej strane nie je ťažké vylepšiť ju (napríklad kúpou jednej vhodnej karty akcií).

Poradie, v akom skúša kupovať karty:

- karta *Province*,
- karta *Duchy*, ak je v zásobe najviac 5 kariet *Province*,
- karta *Estate*, ak sú v zásobe najviac 2 karty *Province*,
- karta *Gold*,
- karta *Silver*.

Pri vyhadzovaní kariet z ruky do odhadzovacej kôpky stratégia postupuje od kariet s najmenšou peňažnou hodnotou k tým hodnotnejším.

Simuláciou 10 000 hier dvoch hráčov používajúcich stratégiu `BigMoney` sme zistili, že prvý hráč sa priemerne dostane na ťah 20.7-krát.

5.2.3 Trieda `Human`

Táto trieda umožňuje ľudským hráčom odohrať hru s počítačovými stratégiami. Na všetky podnety arbitra reaguje tak, že na štandardný výstup vypíše popis podnetu a zoznam možností a potom načíta zo štandardného vstupu odpoveď.

Pre ľudského hráča je prirodzenejšie dostávať informácie o protihráčových ťahoch postupne ako prebiehajú, nie v sumárnej podobe. Napríklad fakt, že si súper kúpil kartu *Province*, sa dá odvodiť porovnaním obsahu jeho kôpky `total` pred a po ťahu. Pre človeka by však bolo jednoduchšie dostať túto informáciu priamo.

Spolu so stratégiou `Human` by sa mal preto používať objekt `VerbosePlayerState`, ktorý na rozdiel od `PlayerState` vypisuje všetky udalosti na štandardný výstup.

Kapitola 6

Testovanie modelov stratégií

V tejto kapitole popíšeme parametre, ktoré sme pri simulovaní evolučných procesov použili. Potom porovnáme jednotlivé modely stratégií a uvedieme najlepšie inštancie, ktoré sa nám podarilo získať.

6.1 Parametre evolúcie

6.1.1 Fitnes funkcia

Na výpočet fitnes funkcie sme použili referenčnú stratégiu `BigMoney`. Testovaná stratégia s ňou odohrala 100 hier, pričom sa pred každou hrou náhodne určilo poradie, v akom budú ťahať. Primárnou zložkou fitnes funkcie bol potom počet hier, v ktorých testovaná stratégia vyhrala.

Vyšší počet hier by priniesol väčšiu presnosť hodnotenia stratégií, na druhej strane by vyžadoval viac času na ich simulovanie. Použitá hodnota je preto kompromisom.

Takáto fitnes funkcia nedostatočne odlišovala rôzne úrovne slabých stratégií. Trvalo veľmi dlho, kým náhodnými mutáciami vznikla stratégia, ktorá by dokázala vyhrať aspoň jednu hru a tak si zaručiť postup do ďalšej generácie. Na úrovni fitnesu 0 teda nebolo možné zlepšovať riešenia postupne.[1]

Preto sme pridali sekundárnu zložku fitnes funkcie, ktorá v prípade rovnosti primárnej zložky porovnáva stratégie podľa počtu víťazných bodov, ktoré v hrách získali.

Jedinca s najvyšším fitnesom v generácii sme potom nechali odohrať 10 000 hier a tak získali presnejšie vyčíslenie jeho kvality. Hodnoty fitnesu použité v nasledujúcich častiach kapitoly a v grafoch pochádzajú práve z tohto merania.

[1] Tento problém sa týkal len referenčnej stratégie `BigMoney` – keď sme použili `GreedyStrategy`, aj jedince z prvej generácie dokázali vyhrať nenulový počet hier. Cenou za to však bola horšia schopnosť odlišovať rôzne úrovne silných stratégií.

6.1.2 Populácia

Za veľkosť populácie sme zvolili $\mu = 5$, z každého jedinca sa vytvorilo $\lambda = 10$ mutantov (tempo mutácie 0.5, sila mutácie 1). Do novej populácie sme vybrali μ jedincov s najvyššou hodnotou fitnes funkcie, pričom sme uvažovali aj riešenia z rodičovskej generácie. Použili sme teda výber $(5 + 10)$.

Ako počiatočnú generáciu sme nevygenerovali náhodné stratégie, ale prázdne – každé z hodnotení kariet malo všetky karty označené ako nepoužité. Terminálnou podmienkou bolo dosiahnutie 100 generácií.

6.2 Testované modely

Väčšina testov prebehla v prostredí základnej sady desiatich kráľovských kariet, ktorá je v pravidlách Dominionu uvedená pod názvom First Game. Model stratégie, ktorý v tomto prostredí dopadol najlepšie, sme potom otestovali aj s ostatnými desaticami z pravidiel.

Všetky testované stratégie boli typu `ExtremalStrategy`. V kontexte `gaining` sme vyskúšali ako hodnotenie zvoliť `WeightedRandomChooser`, `RanksChooser`, `RatiosChooser` a `SeasonalChooser`, ktoré obsahovalo dve hodnotenia s prahom 100 (zodpovedá to približne piatim kartám *Province* v zásobe). Ako prvé z nich sme skúsili použiť všetky tri jednoduché druhy hodnotenia; ako druhé sme vždy použili `RanksChooser`. V kontexte `playing` sme skúšali hodnotenia `RandomChooser` a `RanksChooser`.

Za parameter ℓ sme vo všetkých použitých chromozómoch typu `RatiosChromosome` a `RanksChromosome` zvolili $-\frac{\ln 2}{4}$.

Takto sme dostali $(1 + 1 + 1 + 3) \cdot 2 = 12$ rôznych modelov stratégií. Pre každý model sme vykonali 10 nezávislých simulácií evolúcie.

6.3 Výsledky testovania

Nasledujúce grafy zobrazujú pomer hier, ktoré vyhral najlepší jedinec v jednotlivých generáciách (teda normovanú primárnu zložku fitnes funkcie).

Každá lomená čiara predstavuje jednu simuláciu. Z pôvodných desiatich simulácií sme v grafoch použili kvôli prehľadnosti vždy len niektoré (reprezentatívne).

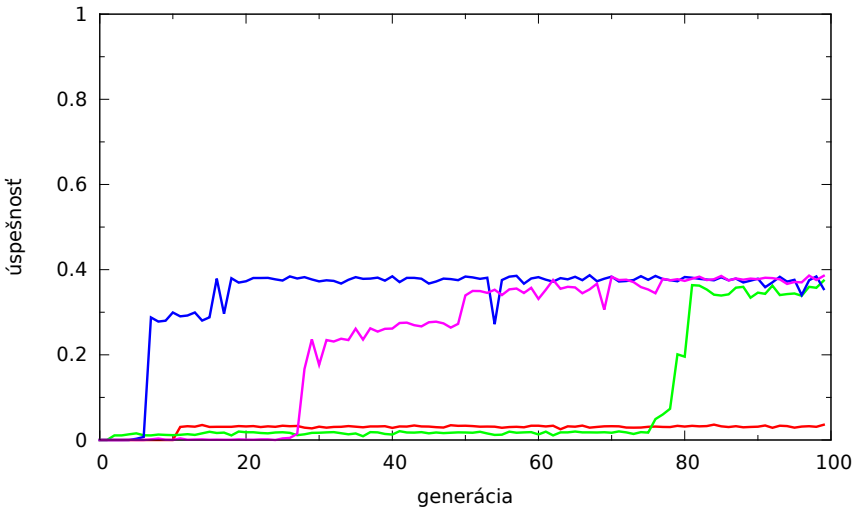
Graf 6.1 zobrazuje priebeh evolúcie modelu stratégie, v ktorom sme v kontexte `gaining` použili hodnotenie kariet `RanksChooser`, no poradie hrania kariet sme nechali na náhode.

Vidíme, že maximálna dosiahnutá úspešnosť sa pohybuje pod hodnotou 0.4 a tiež lokálne maximá okolo úrovní 0.25 a 0.03 (z ktorého sme sa počas jednej zo simulácií nedokázali dostať).

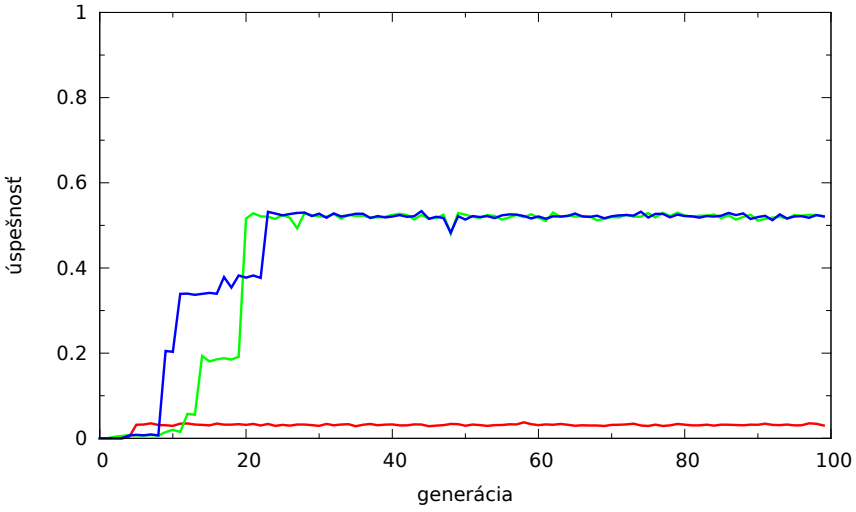
Všimnime si tiež, že jedna zo simulácií dokázala dosiahnuť celkové maximum už v 20-tej generácii.

Príklad genómu získanej stratégie s úspešnosťou 0.38: {`gaining: [Province, Gold, Market, Militia, Silver]`}.

Keď sme k tomuto modelu pridali v kontexte `playing` hodnotenie kariet `RanksChooser`, dostali sme o čosi lepšie výsledky (graf 6.2). Najúspešnejšia stratégia dosiahla hodnotu 0.52.



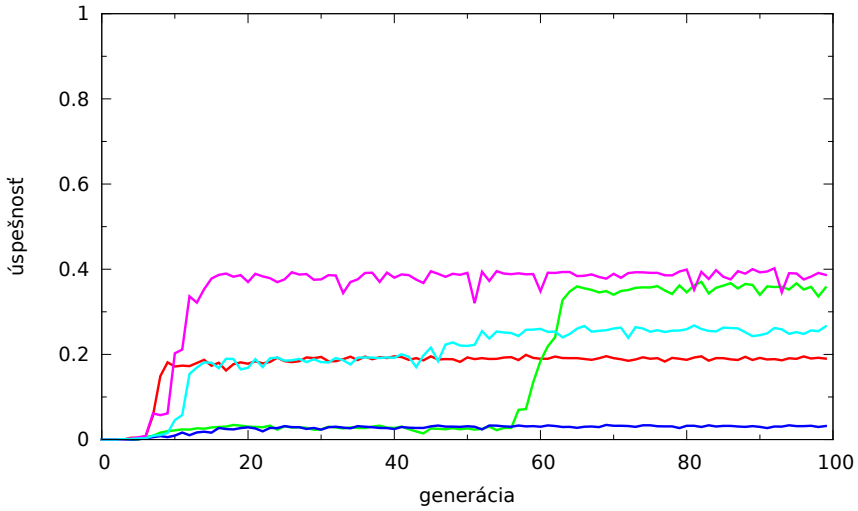
Obr. 6.1: `gaining`: `RanksChooser`, `playing`: `RandomChooser`



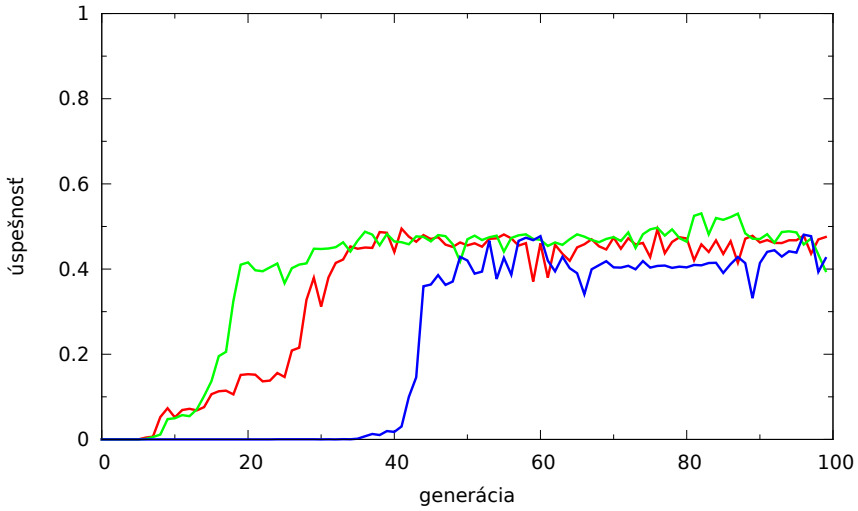
Obr. 6.2: `gaining`: `RanksChooser`, `playing`: `RanksChooser`

Toto je mierne prekvapivé – nečakali sme, že sa inteligentnejšie hranie kariet takto markantne prejaví v kvalite hráča.

Príklad genómu získanej stratégie s úspešnosťou 0.52: {`gaining: [Province, Gold, Market, Militia, Silver], playing: [Estate, Woodcutter, Moat, Market, Gold, Militia]`}. V tomto prípade zrejme rozdiel spôsobilo cielené hranie karty *Market* pred kartou *Militia*.



Obr. 6.3: `gaining`: `WeightedRandomChooser`, `playing`: `RanksChooser`



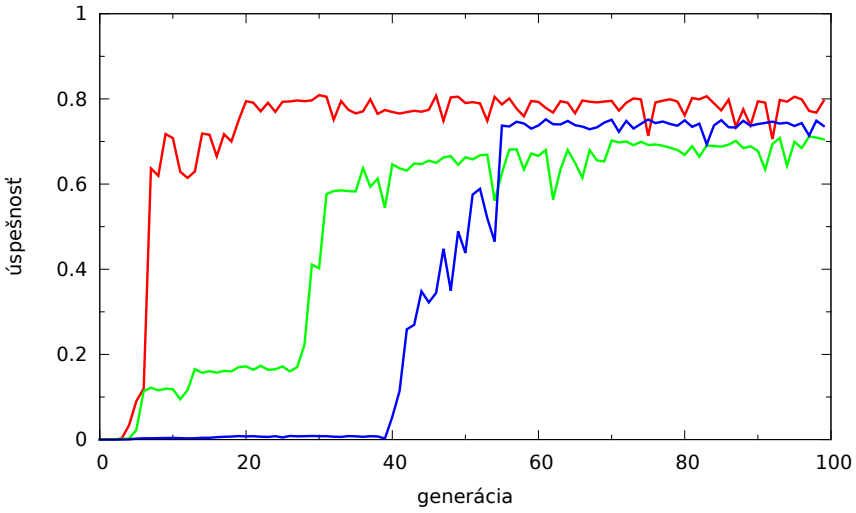
Obr. 6.4: `gaining`: `RatiosChooser`, `playing`: `RanksChooser`

Podobne (len o niečo menej výrazne) sa pridaním `RanksChooser` v kontexte `playing` zlepšili výsledky aj modelom s hodnotením `WeightedRandomChooser` príp. `RatiosChooser` v kontexte `gaining` (grafy 6.3, 6.4).

Z hľadiska stability dopadol zatiaľ najlepšie model na grafe 6.4: len jedna z 10 simulácií nedokázala ani po 100 generáciách prekonať hranicu 0.35; zvyšných 9 to zvládlo do 50-tej generácie; 8 z nich dokonca do 25-tej.

Príklad genómu získanej stratégie (`WeightedRandomChooser`) s úspešnosťou 0.39: {`gaining: {Province: 1.00e+00, Gold: 2.62e-04, Silver: 6.58e-07, Smithy: 5.14e-07}, playing: [Mine, Duchy]`}. Takýto obsah chromozómu `playing` zjavne nemá na hru žiaden vplyv (správa sa ako `RandomChooser`).

Príklad genómu získanej stratégie (`RatiosChooser`) s úspešnosťou 0.47: {`gaining: {Province: 3.83e-01, Gold: 2.64e-01, Militia: 1.55e-01, Silver: 1.17e-01, Market: 8.03e-02}, playing: [Estate, Market, Copper]`}.



Obr. 6.5: `gaining: SeasonalChooser` (`RatiosChooser`, `RanksChooser`), `playing: RanksChooser`

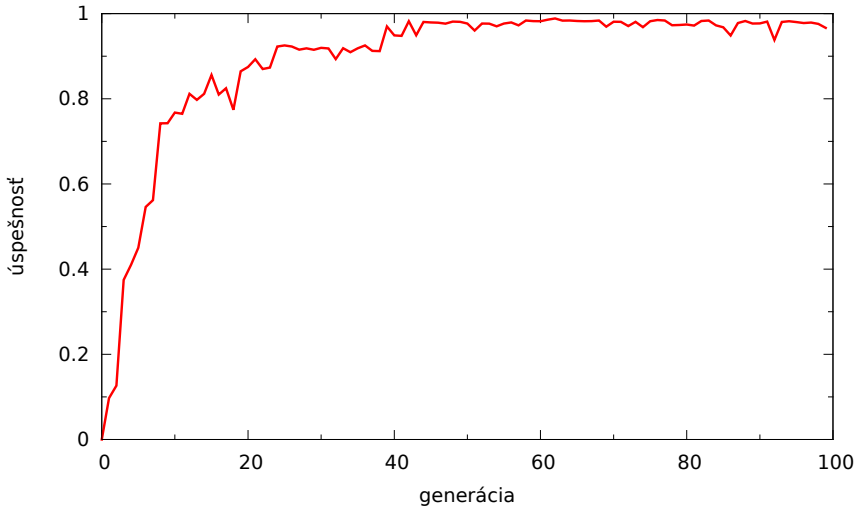
Najlepšie zo všetkých modelov dopadol `SeasonalChooser` s komponentmi `RatiosChooser` a `RanksChooser` (graf 6.5). Aj ostatné kombinácie komponentov však výrazne prekonali svoje jednoduchšie náprotivky. Možnosť zmeniť svoje správanie po začiatočnej fáze hry teda podstatne pridáva modelu stratégie silu.

Príklad genómu získanej stratégie s úspešnosťou 0.80: {`gaining1: {Gold: 3.13e-01, Market: 2.18e-01, Province: 1.93e-01, Militia: 1.46e-01, Silver: 1.30e-01}, gaining2: [Province, Duchy, Gold, Market, Mine, Estate, Militia, Remodel, Village], playing: [Gold, Market, Estate, Militia, Province, Village]`}.

Túto stratégiu sme otestovali aj v desiatich hrách s ľudským protihráčom. Počítačový hráč z toho dokázal vyhrať dve hry a jednu remízovať.

V ostatných prostrediach (desaticiach kráľovských kariet) dosiahol tento model podobné alebo mierne horšie výsledky. Najmenej sa mu darilo porážať referenčnú stratégiu `BigMoney` v sade kariet príhodne nazvanej Big Money (0.59). Pozoruhodný je prípad sady kariet Size Distortion (graf 6.6).

Príklad genómu získanej stratégie s úspešnosťou 0.98: {`gaining1: {Thief: 4.84e-01, Village: 3.33e-01, Gold: 1.83e-01}, gaining2: [Duchy, Estate,`



Obr. 6.6: sada Size Distortion, `gaining`: `SeasonalChooser` (`RatiosChooser`, `RanksChooser`), `playing`: `RanksChooser`

`Gardens, Gold, Workshop, Silver, Copper], playing: [Witch, Province, Duchy, Gardens, Village, Estate]`}.

Záver

V tejto práci sme implementovali prostredie hry Dominion a vytvorili niekoľko stratégií pre všetkých päť variantov hry uvedených v pravidlách.

Evolučné algoritmy sa osvedčili pri hľadaní parametrov predstavených modelov stratégií. Jednotlivé modely sa ukázali ako rôzne silné – ďalšie zvýšenie kvality automatického hráča vyžaduje použitie nových modelov.

Najlepšia vygenerovaná stratégia dokázala vyhrať dve z desiatich hier s ľudským protihráčom a jednu z nich remízovať. Veríme však, že hranica možností automatických hráčov leží omnoho ďalej.

Neskúšali sme meniť parametre evolúcie, je preto možné, že sa dá dosiahnuť rýchlejšia konvergencia a stabilita. V našom prípade sa totiž pre rýchlejšie nájdenie dostatočne kvalitného riešenia oplatilo použiť menší počet generácií a viaceré nezávislé simulácie evolúcie. V praxi by sme však potrebovali ešte efektívnejšiu metódu generovania stratégie.

Ďalším vylepšením by bolo rozšírenie množiny referenčných hráčov. Presvedčili sme sa totiž, že hrou so silnejšími hráčmi vznikajú kvalitnejšie stratégie.

Pre zjednodušenie komunikácie medzi ľudským a počítačovým hráčom by bolo tiež vhodné implementovať grafické používateľské prostredie alebo umožniť počítačovému hráčovi pripojiť sa na už existujúci server s cudzou implementáciou hry Dominion.

Literatúra

[1] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evol. Comput.*, 1(1):1–23, Mar. 1993.

[2] H. Beyer. *The Theory of Evolution Strategies*. Natural Computing Series. Springer, 2001.

[3] C. Darwin. *On the Origin of Species by Means of Natural Selection: Or the Preservation of Favoured Races in the Struggle for Life*. D. Appleton and company, 1869.

[4] M. Dresher. *The Mathematics of Games of Strategy: Theory and Applications*. Dover Books on Mathematics Series. Dover, 1981.

[5] D. Fudenberg and J. Tirole. *Game Theory*. Mit Press, 1991.

[6] Garion and Verik. Big money. http://simulatedominion.wordpress.com/strategies/big-money/.

[7] M. Osborne and A. Rubinstein. *A Course in Game Theory*. Mit Press, 1994.

[8] V. Putman, D. X. Vaccarino, and D. Yu. Dominion (game rules), 2008.

[9] E. Rasmusen. *Games and Information: An Introduction to Game Theory*. Blackwell, 2001.

[10] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2010.