Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Dokumentace k semestrální práci z předmětu
KIV/UIR

Karetní hra sedma

Martin Červenka A14B0239P
cervemar@students.zcu.cz
26. dubna 2016
Počet hodin: 25

Obsah

1 Zadání

Zadáním semestrální práce je naprogramovat karetní hru sedma.

1.1 Pravidla

Zjednodušený popis pravidel dle[1]:

1.1.1 Karty

32listové, nejsilnějším listem hry jsou sedmy. Přebíjejí kteroukoliv kartu. Přitom zde barvy nehrají prakticky žádnou roli, stejně tak tu odpadá pořadí hodnot. Vyšší karta se proti nižšímu listu nijak neuplatní. Přebíjet může nanejvýš karta stejné hodnoty. Kluk zabíjí kluka, osma osmu. Univerzální hodnotou jsou už zmíněné sedmy. Jediné karty, které přnášejí vítězství a přiměřené bodové ohodnocení, jsou desítky a esa, tzv. hodnotné listy.

1.1.2 Cíl hry

Získat co nejvíce zdvihů obsahujících esa a desítky. Kromě toho získat body za poslední zdvih ve hře.

1.1.3 Rozdávání

Rozdá se po čtyřech listech, zbývající karty se jako talón položí na stůl. Odsud si hráči po každém zdvihu dokupují, aby doplnili stav listů v ruce na původní počet.

1.1.4 Průběh hry

Předák otevírá hru. Protože se vzájemně přebíjejí jen karty stejné hodnoty, je účelné vynést od „dvojáka“, tj. kartou, kterou máme zastoupenou alespoň dvakrát. Barvu netřeba ctít, ale není zapotřebí ani přebíjet. Každý hráč je povinen odhodit kartu v libovolné barvě nebo libovolné hodnotě. Sedmy většinou držíme v ruce tak dlouho, dokud nedojde k boji o hodnotné listy - esa a desítky.

1.1.5 Bodování

Po skončení sehrávky spočítají partneři své desítky a esa. Každá z těchto karet představuje 10 bodů stejně jako poslední zdvih. Maximálně dosažitený zisk má takto dohromady hodnotu 90 bodů. Vyhrává hráč, který po dlouhé sérii her získal nejvyšší bodový náskok.

1.2 Upravená pravidla

Do těchto pravidel jsem přidal některá další, která dělají hru atraktivnější.

1.2.1 Hra pro tři hráče

Hru lze modifikovat pro tři hráče. Jediným problémem je 32 karet v balíčku – 32 není dělitelné třemi. Problém lze vyřešit odebráním 2 nevýznamných karet. Řekněme například, že odebereme před rozdáváním dvě karty hodnoty 8.

1.2.2 Spálená hra

Alternativním cílem hry může být, když hráč má v ruce 4 karty stejné hodnoty, pak automaticky vyhrává. Stejně tak když se ocitnou 4 karty stejné hodnoty v těsném sledu v jednom zdvihu na stole – pak vítězí hráč, který položil poslední kartu stejné hodnoty jako měly 3 karty předcházející.

2 Analýza problému

Probém lze rozdělit na dva velké podproblémy. Těmi jsou návrh umělé inteligence a návrh algoritmu ovádající průběh hry.

Pro umělou inteligenci lze použít mnoho metod. Mezi ně patří například procházení úplným stavovým stromem s použitím například **Min-Max** algoritmu. Další možností je pouvžití **genetického** algoritmu. Poslední možnost je navrhnout algoritmus **vlastní**.

Druhým probémem je návrh algoritmu ovládající průběh hry. Konkrétním problémem je, jak vyřešit problém dodržování pravidel při vynášení karty. Zde mám dvě možnosti řešení – buď donutím automatické hráče aby hráli dle pravidel a pravidla ošetřím pouze u lidských hráčů, nebo vytvořím jakousi režii, která každou kartu schválí nebo zamítne.

3 Návrh řešení

Nyní pro problémy uvedené v sekci 2 navrhnu řešení.

3.1 Umělá inteligence

Musím navrhnout, jakým způsobem bude hrát automatický hráč. Zde mám na výběr z několika možností.

3.1.1 Úplný stavový strom

Tahy bych mohl reprezentovat stavovým stromem. Provedu tedy analýzu této metody.

Hra sestává z **x** karet náhodného pořadí, každý hráč obdrží **k** karet. Uvažujme proměnnou **h** jako počet hráčů – pak počet možností rozdávky σ činí[1]:

$$\sigma_u = \frac{x!}{(x - k \cdot h)! \cdot k!^p} \tag{1}$$

Po dosazení $x = 32$, $k = 4$, $h = 2$ získáme poćet možností rozdání karet:

$$\sigma_u = \frac{x!}{(x - k \cdot p)! \cdot k!^p} = \frac{32!}{(32 - 4 \cdot 2)! \cdot 4!^2} = \frac{32!}{24! \cdot 24^2} = 736281000 \tag{2}$$

Asi si lze představit, jak by narůstala časová i paměťová složitost, proto nelze úplný stavový strom použít.

3.1.2 Redukovaný stavový strom

Vyjdeme z předpokladu v 3.1.1, uvědomíme si však, že barvy v této hře nehrají žádnou roli. Poté lze vzorec (1) modifikovat na[2]:

$$\sigma_r = \frac{\frac{x}{k}!^k}{(\frac{x}{k} - h)!^k k!^h} = \frac{\frac{32}{4}!^4}{(\frac{32}{4} - 2)!^4 4!^2} \doteq 17074 \tag{3}$$

I toto číslo je velmi vysoké a to se jedná pouze o rozdávku, proto tento způsob nepoužiji jako řešení.

3.1.3 Genetický algoritmus

Problém by mohl být řešen i genetickým algoritmem. Vzhledem ke komplexnosti problému (viz vzorec (2) v sekci 3.1.1) by potřeboval genetický algoritmus velmi mnoho generací, aby se hru naučil. Druhým problémem je, že by se naučil jak hrát proti jednomu hráči, ale jiný hráč může hrát úplně jiným způsobem. Proto tento způsob nezvolím.

[1] Vyjádření tohoto vzorce naleznete v příloze A
[2] Vyjádření tohoto vzorce naleznete v příloze B

3.1.4 Napodobení člověka

Jako nejvhodnější řešení tedy zvolím naprogramování umělé inteligence s pomocí pravděpodobnosti – na podobném principu hraje většinou i člověk. Na názorné ukázce tuto myšlenku vysvětlím. Hra se zatím vyvynula dle tabulky 1.

Počet hráčů	3
Počet odehraných es	2
Počet odehraných sedem	1
Počet es v mojí ruce	1
Počet sedem v mojí ruce	1

Tabulka 1: Příklad vývoje hry

Nyní jsem na výnosu a přemýšlím, zda-li mám naději úspěšně odehrát své eso. Je zřejmé, že ostatní hráč mohou mít maximálně 1 eso a 2 sedmy – tedy 3 karty na přebíjení. Já mám karty 2. Protiháči jsou dva – pokud je jisté, že všechny důležité karty již protihráči mají, pak určitě prohraji. Jinak ale mám nějakou šanci na výhru. Volba je na tom, zda-li zariskuji či nikoliv.

Navrhnu tedy program, který provede podobnou úvahu jako jsem zde uvedl, a pak dle pravděpodobnosti rozhodne, zda-li zariskuje nebo ne.

Další výsadou lidského hráče je klamání (tzv. blafování). Počítač naučím klamat tak, že i na běžný tah může zahrát s určitou pravděpodobností tah hodnotnou kartou či sedmou, aby klamal ostatní hráče o tom, co má v ruce. Taktéž občas vynese hodnotnou kartu, i když k ní nemá žádné další karty.

3.2 Hra

Každý hráč si může ve svém tahu zvolit kartu a tu odehrát. Jenže musí při tom dodržovat pravidla. Automatického hráče lze naprogramovat tak, aby podle těchto pravidel hrál, ale člověka nedonutíme, aby je dodržoval, pokud nevytvoříme nějaký schvalovací automat. Tento automat bude ověřovat, zda-li je právě zvoléná karta hratelná dle pravidel a hra může pokračovat, nebo zda-li musíme donutit hráče zvolit jinou kartu.

Kromě algoritmů umělé inteligence je třeba naprogramovat i tento schvalovací automat.

4 Popis řešení

Implementoval jsem program dle návrhu řešení. Nejprve jsem naimplementoval logiku akceptačního automatu, který kontroluje pravidla hry a poté jsem vytvořil automatického hráče, kterého jsem se snažil připodobnit člověku.

4.1 Zvolená platforma

Zvolil jsem pro implementaci programovací jazyk C++. Při vývoji aplikace jsem se soustředil na cílovou platformu OS Linux. Aplikace se ovládá pouze s terminálu.

4.2 Třídy

Třídy jsou řádné okomentovány, zde pouze stručně popíši k čemu slouží.

4.2.1 AlgoNo

Třída reprezentuje hloupého robota, který hraje pouze tak, aby dodržoval pravidla. Lepší algoritmus robota je **ProgrammerBot** (viz 4.2.8). Tento algoritmus vzhledem k existenci lepšího algoritmu nemá kromě testování praktický význam.

4.2.2 Algorithm

Abstraktní třída **Algorithm** reprezentuje abstarktního hráče. Od této třídy dědí například třídy **AlgoNo**, **Person** nebo **ProgrammerBot**. Lze provést vlastní implementaci této třídy a tím vytovrit libovolného hráče (složitější UI, člověka ovládajícího aplikaci např. z GUI, síťového hráče atd.) – postup viz 4.3.1.

4.2.3 Card

Třída představující jednu kartu hry, Karta může mít 4 barvy a 8 hodnot.

4.2.4 Deck

Třída **Deck** popisuje chování karet v balíčku. Implementuje metody pro výběr a vložení karty do balíčku, promíchání balíčku a další.

4.2.5 Game

Game je třída definující pravidla hry a stará se o jejich dodržování. V této třídě je popsán postup celé hry od rozdání až po ukončení hry.

4.2.6 Hand

Třída **Hand** představuje ruku hráče, ve které má karty. Tato třída je implementačně pouze pole definované velikosti s několika metodamy obsluhy.

4.2.7 Person

Person je třída, pomocí které je možno hru ovládat. Tato třída je silně platformě závislá na platformě popsané v 4.1. Problém však lze řešit postupem uvedeným v 4.3.2.

4.2.8 ProgrammerBot

Hlavní inteligence programu se skrývá ve třídě **ProgrammerBot**. Je navržena dle podrobného návrhu v sekci 3.1.

4.2.9 Předdefinované chování mimo UNIXový systém

V souboru **stdmcr.h** je nastaveno chování pro neUNIXové systémy. Toto chování je však tak abstraktní, že je v podstatě nepoužitelné. Pro jiné platformy je tedy třeba chování doplnit.

4.3 Snadná rozšiřitelnost

Program jsem navrhl tak, aby byl snadno rozšiřitelný. Hlavní věc, na kterou jsem se soustředil, je možnost doprogramování dalšího algoritmu hráče. Také jsem předpokládal při návrhu, že by v budoucnu někdo chtěl používat program i na jiné platformě než **Linux/bash**, nebo že si dá někdo v budoucnu práci s vytvořením grafického uživatelského rozhraní (GUI).

4.3.1 Přidání vlastního algoritmu hráče

Pro přidání algoritmu hráče do programu je potřeba provést následující úkony:

1. Implementace třídy která dědí od třídy **Algorithm**. Zde je třeba naimplementovat alespoň dvě metody:
   - `Card* Play(bool);`[3]
     Tato metoda slouží pro výběr karty dle algoritmu.
   - `void Used(Card*,unsigned char);`
     Metoda přijme parametry jakou kartu kdo hrál, aby podle toho mohl algoritmus hrát.
2. Přidat zdrojový soubor k překladu. V případě překladu nástrojem **CMake** to lze provést editací souboru **CMakeLists.txt**, kam se přidá do proměnné **SRCS** jméno/cesta k zdrojovému souboru nového algoritmu.
3. Přidat algoritmus do algoritmů. To se provede editací souboru **config.cxx**. Zde se dopíše do metody **GetAlgorithm** další větev `else CMP`, při editaci si lze vzít příklad z již existujících větví.
4. V adresáři se zdrojovými soubory provedeme sekvenci příkazů:
   ```
   cd build
   cmake ..
   make
   ```
5. Vložení nového hráče do souboru **config.txt** (jak editovat viz 5.2).

[3] Podrobnosti viz programová dokumentace

4.3.2 Platformní závislost

Jak jsem již uvedl, program silně závisí na zvolené platformě. V případě použití nekompatibilní platformy lze modifikovat soubor **stdmcr.h** který je k tomuto účelu vytvořen.

4.3.3 Vytvoření GUI

V hlavičkovém souboru **stdmcr.h** lze nastavit reakce na výpis na obrazovku, použití barev atd. Tím lze řešit nejen platformní závislost ale lze zde doimplementovat i logiku která bude tyto zprávy zachytávat a předávat do GUI. Tato logika však není součástí mojí implementace.

5 Uživatelská dokumentace

5.1 Popis sestavení programu

Program se sestaví do složky **build**. V této složce se již nachází předpřipravený soubor **config.txt**. Sestavení programu provedeme následující sekvencí příkazů:

- `cd build`
- `cmake ..`
- `make`

V případě úspěšného překladu vznikne spustitelný soubor **cards**.

5.2 Nastavení programu

Pro spuštění programu je třeba mít správně nastavenou hru. Toto nastavení se provede v souboru **config.txt**. V tomto souboru se nachází:

- `NumberOfPlayers = x`
  Kde x je celočíselná hodnota od 2 do 8[4]
- `x` následujících dvojic řádek
  - `Name = y`
    Kde y je alfanumerický řetězec neobsahující mezery.
  - `Player = z`
    Kde z je jméno třídy která se pro tohoto hráče použije.
    /`Person`, `ProgrammerBot`, `AlgoNo`/
- Nepovinná řádka `OwnRules` která aktivuje pravidla ze sekce 1.2

[4] Rozumné hodnoty jsou od 2,3 nebo 4

5.3 Ovládací panel

Program spustíme příkazem `./cards`. Pro správné spuštění je třeba, aby se v pracovní složce nacházel soubor **config.txt** – nastavení hry. Po spuštění se zobrazí panel na obrázku 1.

Obrázek 1: Spuštění hry

Znak zavináče značí kurzor. Tím lze pohybovat šipkami. Po stisku pravé šipky se například kurzor přesune na 2. pozici jak zobrazuje obrázek 2.

Obrázek 2: Posun kurzoru

Stiskem tlačítka `ENTER` vybereme kartu, a pokud to pravidla (viz 1.1) dovolují, karta bude odehrána. Stav po této akci zachycuje obrázek 3.

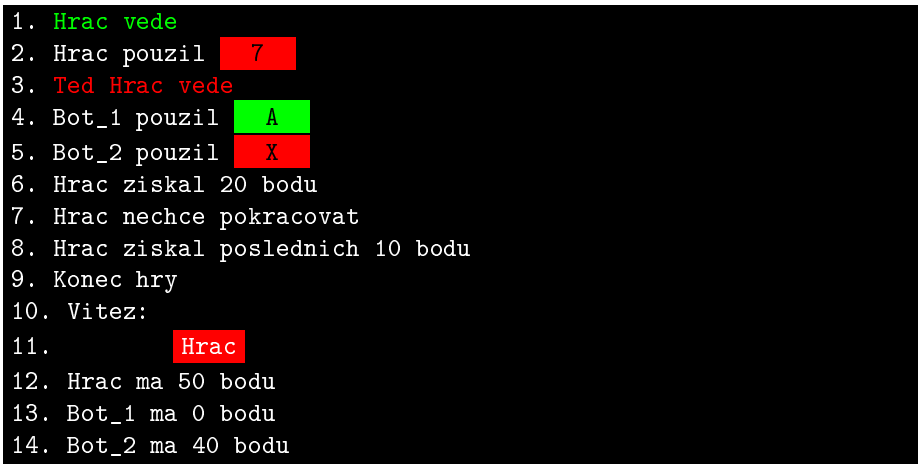
Obrázek 3: Odehrání karty

Pokud je třeba zahrát tah „nechci pokračovat“, pak lze kurzor posunout až za karty, zobrazí se znak šipky. Následným stiskem tlačítka `ENTER` akci potvrdíme. Situaci ilustruje obrázek 4.

Obrázek 4: Tah „nechci pokračovat“

5.4 Informační panel

V informačním panelu se zobrazují údaje o tom, kdo jakou kartu zahrál, kdo získal kolik bodů, kdo aktuálně vede a na konci hry kdo vyhrál. Typickou situaci konce jedné hry zachycuje následující obrázek 5.



Obrázek 5: Informační panel

Na obrázku 5 je vidět poslední sehrávka hry a vyhodnocení. Na tahu je **Hrac**. To značí zeleně vypsaná řádka 1. **Hrac** si vybral kartu (jak vybrat kartu viz sekce 5.3) – červenou sedmu. Výnosem tedy vede **Hrac** což potvrzuje řádka 3. Protože je poslední zdvih, **Bot_1** i **Bot_2** musejí odehrát své poslední karty. Těmito kartami jsou zelené eso a červená desítka. To je pro **Hrac**e výhodné – získá tím 20 bodů (řádka 6). **Hrac** nechce pokračovat – on už ani pokračovat nemůže. Za poslední zdvih ve hře získá **Hrac** dalších 10 bodů.

Nyní vyhodnocení celé hry – celou hru vyhrál **Hrac**; získal 50 bodů. Těsně za ním skončil **Bot_2** se 40 body. **Bot_1** je beznadějně poslední s 0 body.

5.5 Další hra

Na konci hry se zobrazí výzva jako na obrázku 6.

Obrázek 6: Dotaz na další hru

Pokud stiskneme jakýkoliv alfanumerický znak a **ENTER**, pak bude pokračovat další hra. Stiskem tlačítka **n** a **ENTER** se hra ukončí. Hru lze ukončit i stiskem kláves **Ctrl**+**C**.

6 Závěr

Po naprogramování jsem hru mnohokrát otestoval. Obtížnost protiháčů se mi zdá uspokojivá. Pro profesionální hráče této hry možná bude tato obtížnost lehká. Nejvíce zde stejně závisí na náhodě. Při hraní proti hráči lze vždy trochu vidět, co bude hrát, zda-li rád riskuje atd. Jenže zde nelze předpokládat nic.

Podařilo se mi tedy naprogramovat hru, která je sice hratelná, ale se současným algoritmem umělé inteligence si hráč asi moc hru neužije.

Reference

[1] Pravidla hry sedmy:
Omasta V., Ravik S., Karty, hráči, karetní hry, 80-7309-206-9, str. 143-145

A Vyjádření vzorce σ_c

A.1 Úloha

Nechť existuje balíček **x** karet. Z těchto karet vybírá **h** hráčů **k** karet. Na pořadí karet v ruce ani ve zbylých kartách nezávisí. Jaký je počet možností výběru karet.

A.2 Řešení

Hráč s indexem 0 si vybere z balíčku **x** karet **k** karet.

$$a_0 = \binom{x}{k} \tag{4}$$

Na hráče s inedxem 1 zbyde o **k** karet méně.

$$a_1 = \binom{x-k}{k} \tag{5}$$

Na hráče s indexem **y** zbyde o **y**·**k** karet méně.

$$a_y = \binom{x-y\cdot k}{k} \tag{6}$$

Všechny pravděpodobnosti nyní vynásobíme.

$$\sigma_u = \binom{x}{k} \cdot \binom{x-k}{k} \cdot \ldots \cdot \binom{x-h\cdot k}{k} \tag{7}$$

Tudíž základní vzorec bude:

$$\sigma_u = \prod_{a=0}^{h-1} \binom{x-a\cdot k}{k} \tag{8}$$

Tento vzorec však lze zjednodušit, protože u dvou sousedních členů se vždy zkrátí faktoriály.

$$\sigma_u = \prod_{a=0}^{h-1} \binom{x-a\cdot k}{k} = \prod_{a=0}^{h-1} \frac{(x-a\cdot k)!}{k!\cdot((x-a\cdot k)-k)!} = \prod_{a=0}^{h-1} \frac{(x-a\cdot k)!}{k!\cdot(x-(a+1)\cdot k)!} \tag{9}$$

Zde je vidět že se ve zlomku budou krátit $(x-a\cdot k)$ s $x-(a+1)\cdot k$ z předchozího členu. Potřebujeme tedy určit krajní členy, které zbydou:

$$(x-a\cdot k)!, a=0 => x! \wedge (x-(a+1)\cdot k)!, a=h-1 => (x-h\cdot k)! \tag{10}$$

Ve jmenovateli zbylo stále $k!$, to však nezávisí na a, tím pádem pouze dostane mocninu p. Výsledkem pak bude vzorec:

$$\sigma_u = \frac{x!}{(x-k\cdot h)!\cdot k!^p} \tag{11}$$

B Vyjádření vzorce σ_r

B.1 Úloha

Shodná s A.1, nezáleží však na barě karty.

B.2 Řešení

Každý hráč nyní může vybrat „opakovaně“ kartu dané hodnoty, protože barvu nectíme. Vybereme si tedy z **q** hodnot (s opakováním). Ze vše těchto možností odečteme ty, kde se nachází 2 nebo více karet stejné hodnoty.

$$\frac{q^k}{k!} \tag{12}$$

V nejlepším případě háč vybere karty jedné hodnoty, pak tedy vybíráme z **q-1** hodnot:

$$\frac{(q-1)^k}{k!} \tag{13}$$

Pro posledního háče zbyde **q-h+1** hodnot:

$$\frac{(q-h+1)^k}{k!} \tag{14}$$

Všechny pravděpodobnosti vynásobíme a vznikne vzorec:

$$\prod_{a=0}^{h-1} \frac{(q-a)^k}{k!} \tag{15}$$

Vzorec lze dále upravovat:

$$\prod_{a=0}^{h-1} \frac{(q-a)^k}{k!} = \frac{\prod_{a=0}^{h-1}(q-a)^k}{k!^h} = \frac{(\prod_{a=0}^{h-1}(q-a))^k}{k!^h} = \frac{\frac{q!^k}{(q-h)!^k}}{k!^h} = \frac{q!^k}{(q-h)!^k k!^h} \tag{16}$$

Pokud budeme **q** definovat jako $\frac{x}{k}$ pak výsledný vzorec bude:

$$\sigma_r = \frac{\frac{x}{k}!^k}{(\frac{x}{k}-h)!^k k!^h} \tag{17}$$