MASARYKOVA UNIVERSITA
FAKULTA INFORMATIKY

Scrabble

DIPLOMOVÁ PRÁCE

Bc. Vít Hotárek

Brno, zima 2003

Prohlášení

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

Poděkování

Děkuji RNDr. Pavlu Smržovi, Ph.D. za poskytnutou volnost při realizaci aplikace implementující tuto hru. Rovněž mé poděkování patří RNDr. Petru Kaňovskému za poskytnuté elektronické partie a dále Ing. Petru Kučovi z České asociace Scrabble za informace týkající se pravidel této hry.

Klíčová slova

lexikon, trie, konečný automat, orientovaný acyklický graf, backtracking, heuristika

Shrnutí

Tématem této diplomové práce je studie deskové hry Scrabble [1]. Úkolem bylo seznámit se s nástroji pro práci s českým korektorem pravopisu, které mohou být použity k implementaci prohledávací fáze hry. Dále se bude práce věnovat možnostem efektivního uložení slovníků pro hru a vyhledávání nad těmito strukturami, společně s jejich vzájemným porovnáním. Následující kapitola se pak zaměří na algoritmy, které jsou používány pro generování tahů. V další části práce jsou diskutovány možnosti aplikace různých herních strategií, které berou v úvahu omezení daná počtem dostupných písmen a aktuálním stavem hrací plochy. V praktické části je popsána implementovaná aplikace, zhodnoceno chování programu a jeho výsledky v reálných hrách.

[1] Scrabble® je registrovaná obchodní značka společnosti Milton Bradley, divize Hasbro, Inc.

Obsah

1 **Úvod** . . . 3
2 **Stručně o Scrabble** . . . 4
2.1 Historie . . . 4
2.2 Pravidla . . . 5
2.2.1 Začátek hry . . . 5
2.2.2 Průběh hry . . . 6
2.2.3 Pokládání kamenů . . . 6
2.2.4 Bodování . . . 7
2.2.5 Konec hry . . . 7
2.2.6 Pravidla přípustnosti slov . . . 8
3 **Struktury pro ukládání slovní zásoby** . . . 9
3.1 Naivní přístup . . . 10
3.2 Trie . . . 10
3.3 Konečné automaty . . . 12
3.3.1 Základní pojmy . . . 12
3.3.2 Techniky minimalizace a tvorby MDAKA . . . 13
3.3.3 Jednoduchý MDAKA . . . 14
3.3.4 Moorův stroj . . . 15
3.3.5 Mealyho stroj . . . 16
3.3.6 Porovnání . . . 18
3.3.7 GADDAG . . . 20
4 **Algoritmus pro generování tahů** . . . 22
4.1 Redukce problému . . . 22
4.2 Backtrackingový algoritmus . . . 25
4.2.1 Tvorba prefixů . . . 25
4.2.2 Tvorba sufixů . . . 26
4.2.3 Detaily . . . 28
5 **Herní strategie** . . . 30
5.1 Slovní zásoba . . . 30
5.2 Strategie výběru tahu . . . 31
5.2.1 Hladová strategie . . . 31
5.2.2 Vyhledávací stromy . . . 32

5.2.3 Heuristiky 34
5.3 Výměna kamenů a *PASS*. 35
6 **Implementace** 36
6.1 Specifikace 36
6.2 Použité programové vybavení 37
6.3 Vytvoření slovní zásoby 38
6.4 Realizace 38
6.4.1 Uložení lexikonu 39
6.4.2 Použitý algoritmus 39
6.4.3 Strategie 39
6.5 Praktické výsledky 40
7 **Závěr a směry dalšího vývoje** 44
A **Statistiky** 48
A.1 Statistiky odehraných slov 48
A.2 Statistiky výskytu písmen ve slovech 49
A.3 Ukázka výstupu simulace 50
B **Ukázka grafické podoby programu** 51
C **Podoba hracího plánu** 52

Kapitola 1

Úvod

V poslední době se projevuje stále větší zájem o stolní deskové hry. Kromě klasických zástupců jako jsou šachy a dáma, se objevují nové varianty těchto her a konkrétně v našich zeměpisných šířkách se zvyšuje i dostupnost jiných. Díky tomu se pak do obecného podvědomí dostávají také i zástupci méně tradičních her, jako jsou například *go* nebo *backgammon* (vrhcáby).

Jednou z oblíbených her, ne-li přímo nejoblíbenější, v anglicky mluvících zemích je již po několik desetiletí desková hra *Scrabble*. Její popularita je ohromná a díky jejímu stálému rozšiřování i do jiných zemí dále roste. Oblíbenost této společenské hry spočívá v jednoduchosti pravidel a vhodné kombinaci využití jazykových schopností s prvky strategie.

Cílem teoretické části předkládané práce je provést studii této hry pro její převedení do elektronické podoby. V úvodní kapitole budou pro ilustraci problematiky zmíněna základní pravidla hry, v další části budou diskutovány možnosti ukládání slovní zásoby. V následujícím oddílu bude zevrubně představen fundamentální algoritmus pro generování tahů, spolu se zhodnocením jeho efektivity. Poslední kapitola teoretické části se bude věnovat rozboru možností uplatnění taktického myšlení ve hře.

Úkolem v praktické části bylo aplikovat znalosti získané v teoretické části práce a vytvořit fungující systém implementující Scrabble. V těchto kapitolách budou obsaženy specifikace pro navrhovaný systém a popsána forma realizace. V závěru bylo cílem provést zhodnocení chování programu a výsledků dosažených ve skutečných hrách.

Kapitola 2

Stručně o Scrabble

V této kapitole budou stručně vysvětlena základní pravidla Scrabble, spolu s pojmy, se kterými se bude dále v textu pracovat. Nejprve však krátce k historii této deskové hry (více o historii pak v [CAS]).

2.1 Historie

Kořeny této hry sahají do 30. let minulého století, konkrétně do roku 1931, kdy se v USA v domě architekta Alfreda Moshera Buttse zrodila myšlenka vymyslet strategickou hru, jež měla za úkol procvičit jazykové schopnosti a která nebude pro obyčejného člověka příliš složitá. Koncem roku 1931 byla hotova první verze této hry s názvem: **Lexico**. Lexico se hrálo bez desky a hráči získávali body podle délky slova, které bylo vytvořeno. V roce 1933 se Butts rozhodl hru patentovat, ale všechny jeho tehdejší žádosti byly zamítnuty.

Až v roce 1938 došlo k průlomu ve vývoji hry. A. M. Butts, vášnivý luštitel křížovek, se rozhodl spojit písmena s hrací deskou tak, aby bylo možné slova utvářet podobně jako v křížovkách. Ručně vyrobená deska z této doby měla rozměry, které se uchovaly až do dnešní doby, stejně jako zásobník a hodnoty samotných písmen. Po dalších neúspěších u patentových úřadů se Butts rozhodl opustit myšlenku komerčního prodeje hry (tentokrát pod názvem Criss-Crosswords).

Ke hře se vrátil až po druhé světové válce, kdy se díky jeho příteli, Jamesi Brunotovi a jeho ženě daly věci opět do pohybu. Brunot usoudil, že by hra měla být prodávána na trhu a uzavřel s Buttsem dohodu o volném prodeji. Byly provedeny úpravy pravidel hry (přidána prémiová pole a zjednodušena pravidla), došlo také ke změně názvu hry – poprvé se objevuje název **Scrabble**. Konečně se také podařilo hru patentovat a od té doby je název zaregistrován jako ochranná známka.

Přes počáteční neúspěchy s prodejem hry se dostavily první kladné ohlasy. Ale až v roce 1952 došlo ke zvýšení objemu prodeje poté, co se

hra zalíbila prezidentu tehdejšího největšího světového obchodního domu Macy's Jacku Straussovi, který se okamžitě rozhodl hru prodávat ve velkém. Poptávka rostla a Scrabble se počal rozšiřovat i do ostatních zemí světa, hlavně Británie a Austrálie, kde zaznamenal okamžitý úspěch. V průběhu druhé poloviny 20. století se hra dostala do celého světa.

V roce 1991 se konalo první mistrovství světa ve Scrabble v Londýně. Další šampionát se konal za dva roky v New Yorku. Po mnoha prodejích licenčních práv nakonec práva připadla gigantu na poli hraček, společnosti Milton Bradley. Česká verze hry se poprvé objevila v roce 1993, autorem je Martin Hloušek a první mistrovství republiky se konalo v roce 1994. Česká asociace Scrabble (ČAS) vznikla na jaře 1998 s cílem sdružovat české kluby, pořádat mistrovství ČR a průběžně aktualizovat oficiální slovník.

2.2 Pravidla

Prezentovaná pravidla se týkají české verze Scrabble tak, jak byla schválena sněmem ČAS [CAS] a nebudou zde uvedena v plném znění. Na úvod je vhodné vysvětlit pojmy, které se dále v kapitole vyskytují:

- **sáček s písmeny** – obsahuje hrací kameny, každý hrací kámen obsahuje písmeno a bodové ohodnocení kamene. V klasické verzi Scrabble se dále vyskytují 2 prázdné kameny (**žolík**), které mohou v průběhu hry zastupovat libovolné písmeno. Avšak s nulovým bodovým ohodnocením.
- **zásobník** – přihrádka, do níž se umísťují kameny tažené ze sáčku.
- **hrací deska** – čtvercová plocha standardní velikosti 15×15 polí, na kterou se postupně umisťují hrací kameny. Plán hrací desky je možno naleznout v příloze C.
- **středové pole** – speciální pole ve středu hrací desky, kterým musí procházet první odehrané slovo.
- **prémiové (bonusové pole)** – speciální pole, které bodově zvýhodní písmeno/slovo (viz část 2.2.4 o bodování).

2.2.1 Začátek hry

Losování hráče, který bude hru začínat, se provádí následovně: každý hráč si vytáhne 1 kámen ze sáčku. Hráč s písmenem nejblíže k počátku abecedy

začíná. Dojde-li ke shodě, výběr kamene se opakuje. Poté si oba hráči v určeném pořadí vytáhnou 7 kamenů a uloží do zásobníku. Začínající hráč utvoří ze svých kamenů platné slovo a položí na hrací desku tak, aby slovo procházelo středovým políčkem. Následující hráč je po levici začínajícího.

2.2.2 Průběh hry

Hráči postupně vytvářejí z písmen ze zásobníku a kamenů již na desce slova. Po odehrání si sečtou body za veškerá vytvořená či obměněná slova. Je-li spoluhráči některé ze slov považováno za nepřípustné, mohou ho zpochybnit před zahájením tahu dalšího hráče. Pokud je námitka uznána, hráč si vezme kameny zpět a ztrácí tah. Po úspěšném odehrání písmen na plochu si hráč dolosuje stejným počet kamenů, kolik jich použil při tvorbě slov.

Místo pokládání kamenů může hráč tah využít k výměně svých kamenů. Měněné kameny položí lícem dolů a dobere ze sáčku stejný počet kamenů. Poté položené kameny uloží do sáčku a ten zamíchá. Dále již slova netvoří a pokračuje hráč po levici.

Pokud hráč nechce/nemůže tvořit slova nebo nechce měnit kameny, může se vzdát tahu. To provede oznámením **PASS** ostatním hráčům. Vzdát tahu se hráč v průběhu hry může kdykoliv.

2.2.3 Pokládání kamenů

Alespoň jeden z nově pokládaných kamenů musí sousedit s některým již odehraným kamenem na hrací ploše. Úhlopříčné sousedství je nepřípustné.

Všechny nově položené kamenu musí ležet ve stejném řádku nebo sloupci a nesmí být mezi nimi mezera, tj. musí existovat slovo, na kterém se všechny podílejí.

Nové slovo, vytvořené či obměněné přidáním nových kamenů, musí být čitelné zleva doprava nebo shora dolů a musí být povolené (viz část 2.2.6 o přípustnosti slov). Na všech místech doteku s jinými již položenými kameny musí vznikat povolená slova, čtená buď zleva doprava či shora dolů. Nová slova vznikají tedy přidáním kamenů ke slovu na desce, umístěním slova kolmo k existujícímu nebo položením rovnoběžně vedle odehraného slova, tak aby těsně sousedící kameny tvořily platná slova.

Prázdný kámen lze použít k zastoupení libovolného písmene abecedy, i takového, které se na žádném kameni v sáčku nevyskytuje (např. 'W'). Hráč je povinen oznámit, jaké písmeno žolík zastupuje a toto písmeno je kameni přiřazeno až do konce hry.

2.2.4 Bodování

Hráč získává body za každé písmeno všech nově utvořených nebo obměněných slov, podle hodnoty uvedené na kameni. Počítají se tedy i ty kameny, které již na desce ležely. Písmena ležící ve dvou nových slovech se započítávají opakovaně, včetně prémií.

Na hracím plánu se nacházejí i speciální bonusová pole. Jejich význam je následující:

- světle modrá pole zdvojnásobují hodnotu písmene
- tmavě modrá pole ztrojnásobují hodnotu písmene
- růžová pole zdvojnásobují hodnotu celého slova
- červená pole ztrojnásobují hodnotu celého slova

Prémie se započítávají pouze hráči, který kámen na bonusové pole položil a pouze v jednom tahu. V dalších tazích je počítána pouze základní hodnota kamene.

Je-li na slovním prémiovém poli prázdný kámen, slovní skóre se násobí i přesto, že prázdný kámen nemá žádné bodové ohodnocení.

Jestliže jsou v jednom slově využity současně slovní i písmenné prémie, je slovní bonus započítáván až jako poslední. Využije-li hráč dvě slovní prémie v jediném slově, počítají se postupně obě. Maximálně je tedy možno získat čtyř- nebo devítinásobek bodové hodnoty slova. Zvláštní bonus 50 bodů (ozn. též jako *bingo*) pak obdrží hráč, který v jednom tahu odehraje všech svých sedm kamenů ze zásobníku.

Na konci hry je ze skóre každého z hráčů odečtena hodnota kamenů, které zůstaly v jejich zásobníku. Naopak, pokud některému z hráčů v zásobníku kámen nezbyl, přičtou se k jeho skóre hodnoty všech kamenů, které zbyly ostatním hráčům.

2.2.5 Konec hry

Konec hry nástavá v situaci, kdy některý hráč využil všechny své kameny a nelze již ze sáčku vylosovat další. Nikdo další již nesmí táhnout a skóre všech hráčů se upravuje podle pravidel z přechozí části.

Hra však nekončí, nejsou-li již v sáčku k dispozici žádné kameny a všem hráčům nějaké zbývají. V tomto případě hra končí pouze ve stavu, kdy všichni hráči ve dvou po sobě jdoucích kolech provedou PASS.

2.2.6 Pravidla přípustnosti slov

Pravidla přípustnosti slov pro českou verzi Scrabble jsou obsáhle publikována na webových stránkách ČAS, včetně pramenů pro posuzování platnosti jednotlivých slovních tvarů. Zde budou zmíněna jen nejdůležitější omezení na tvorbu slov.

Následují některé nepřípustné kategorie:

- jednopísmenné výrazy
- vlastní jména (*Afrika, Josef,*. . .)
- citoslovce (*prr, tú, hó,*. . .)
- zkratky (*např., apod., atd.,*. . .)
- značky (*km, hl,*. . .)
- slova zastaralá, řídká, nářeční, vulgární a argotická.

Omezení na tvorbu slov (ohýbání) pro jednotlivé slovní druhy lze shrnout do následujících bodů:

1. **podstatná jména**: 1. pád jednotného a množného čísla.
2. **přídavná jména**: 1. pád jednotného i množného čísla všech rodů ve všech třech stupních.
3. **zájmena**: 1. pád jednotného i množného čísla všech rodů a druhů, kromě stažených tvarů (*tys*).
4. **číslovky**: 1. pád jednotného i množného čísla všech rodů a druhů
5. **slovesa**:
   - infinitiv v obou tvarech (*-t, -ti*)
   - všechny osoby jedn. i množného čísla přítomného i budoucího času.
   - příčestí činné (minulý čas) v jedn. i množném čísle všech rodů.
   - příčestí trpné přechodných sloves v jednotném i množném čísle všech rodů
   - rozkazovací způsob
6. **příslovce**: všechny tři stupně, jen u příslovcí odvozených z přídavných jmen. Záporná příslovce jen uvedená v pramenech.
7. **předložky, spojky a částice**: jen tvary uvedené v pramenech.
8. **citoslovce**: nejsou povolena vůbec.

Kapitola 3

Struktury pro ukládání slovní zásoby

V této části budou představeny používané vyhledávací struktury pro efektivní uložení slovní zásoby. Ty jsou využity v prohledávací fázi, která je popsána v následující kapitole. Souhrnně se pak v literatuře uchovávávaná slovní zásoba označuje jako *lexikon*, aby se odlišil význam termínu „slovník“. Lexikony jsou tedy podmnožinou slovníků a většinou si je možné představit jako seznam slov, která nejsou nijak vzájemně provázána a četnost slov se pohybuje v řádu statisíců až milionů.

Nejčastěji jsou pak taková data využita v aplikacích, jakými jsou například korektory překlepů (angl. spell-checkers), thesaury, nástroje pro zpracování přirozeného jazyka (např. obnova diakritiky v textu) a jiných, ve kterých je jedním z úkolů test na přítomnost slovního tvaru v lexikonu. Navíc, nedochází-li v těchto strukturách k častým změnám, jsou takové lexikony nazývány *statické*.

V případě slovních her je slovní zásoba použita primárně k účelům:

1. Validace slov vytvořených člověkem.
2. Zdroj pro hledání slovních tvarů.

Základními požadavky na datové struktury pro uchování lexikonu jsou následující:

- **Paměťové nároky** – jak již bylo zmíněno, lexikony obsahují extrémní množství slovních tvarů, jejich velikost se může pohybovat v řádech desítek megabajtů. Práce s takto velkým objemem dat v paměti by znamenala nadměrnou paměťovou režii a vysoké nároky na výpočetní sílu. Proto je žádoucí objem dat redukovat s pomocí vhodných a sofistikovaných algoritmických postupů.
- **Rychlost prohledávání** – důležitou vlastností použité vyhledávací struktury je rychlost a způsob jejího procházení. Zvlášte u Scrabble je často (při generování tahů počítače) potřeba přistupovat k velkému

množství slov, což u nevhodně zvolené reprezentace může zapřičinit prodlevy v odezvě programu.

Nyní se bude práce věnovat jednotlivým možným reprezentacím slovní zásoby, které mohou být použity pro deskovou hru Scrabble, spolu s jejich vlastnostmi.

3.1 Naivní přístup

Nejjednodušší reprezentace, která předpokládá lexikon ve tvaru obyčejného souboru se seznamem všech slov, pokud možno setříděným. Přístup do takového souboru je zcela přímočarý a pro rychlou validaci slov (např. binárním vyhledáváním) postačující. Avšak naprosto nevyhovující pro nacházení přípustných slov pro hru a díky absenci jakékoliv komprese i příliš objemný (soubor 1 800 000 slov s délkou 2–15 znaků zabírá cca 20MB diskového prostoru). Daleko lepších výsledků je možné dosáhnout použitím další struktury – *trie*.

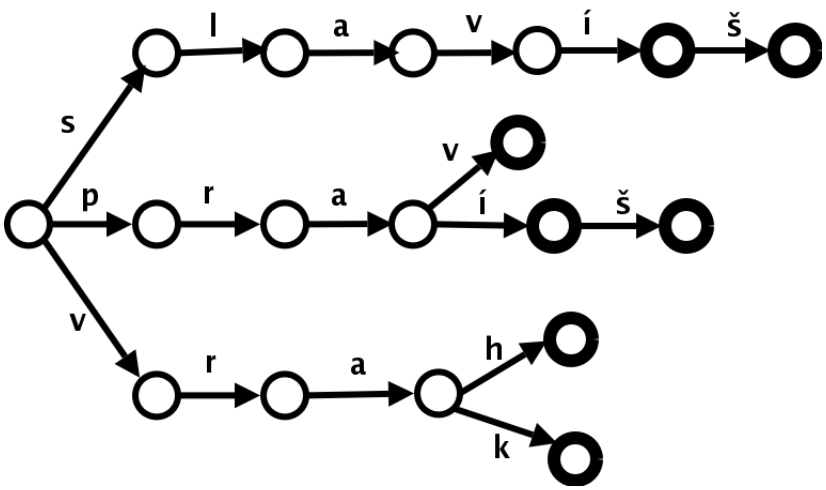
3.2 Trie

Při bližším nahlédnutí na data (zde slovní tvary), která mají být v lexikonech uložena, je možno si všimnout jistých podobností a pravidelností plynoucích z charakteru použitého přirozeného jazyka. Například v lexikonu pro češtinu lze díky skloňování či časování nalézt mnoho tvarů, jež sdílejí společný prefix nebo naopak sufix určité délky, z čehož plyne, že není efektivní ukládat celá slova a je vhodné tyto společné části jistým způsobem ztotožnit.

Problém společných prefixů řeší stromová struktura datovaná svým vznikem do r. 1959 [Bri59], nazvaná *trie* [Fred60]. [1] Nechť M je množina všech slov, která bude takto reprezentována, *klíč* je prvek (tedy slovní tvar) této množiny a Σ označuje abecedu. Klíč se chápe jako posloupnost prvků z množiny Σ. V trie pak každá cesta z kořene do listu odpovídá právě jednomu klíči reprezentované množiny M a výsledný strom je $n-$ární, kde n je rovno $|\Sigma|$. Příklad podoby trie je zachycen na obrázku 3.1.

Z obrázku je patrné, že pokud dvě slova sdílejí společný prefix, je tento prefix uložen pouze jednou a slova poté sdílejí část společné cesty v grafu. Koncové uzly (na obr. 3.1 zobrazeny tučně) jsou nazývány *terminální*. V trie jsou pak všechny listy terminální.

[1] Název je odvozen z angl. „information retrieval."



Obrázek 3.1: Trie

Vyhledávání klíče probíhá přímočaře, nechť T je trie, U je ukazatel do struktury a I pozice znaku v klíči K:

1. $U = Kořen(T)$; Nastav ukazatel na kořen trie
2. $I = 1$; Nastav I na pozici prvního znaku v klíči
3. Není-li v seznamu hran vycházejících z U hrana označená $K[I]$, vrať NENALEZENO a skonči.
4. Proveď průchod po této hraně do uzlu U' a zvyš I o jedna
5. Je-li I rovno $|K|$ a U' je terminální, vrať NALEZENO. Jinak pokračuj bodem 3 pro $U = U'$.

Co se týče použití trie pro generování tahů je situace následující. Hledání tahů se ve velké míře opírá právě o průchod strukturou, s omezeními plynoucími z obsahu zásobníku písmen a již odehranými kameny na hrací desce. V tomto ohledu je trie vhodnou vyhledávací strukturou. Zmíněná omezení slouží k „odřezání“ větví stromu, které prokazatelně nevedou do terminálních uzlů.

Jako příklad je možné vzít následující situaci ve hře – hledají se všechna slova, která lze vytvořit z obsahu zásobníku a začínají jistým prefixem (tento

je již odehrán na ploše). Provede se průchod stromem pro daný prefix a z výsledného uzlu se pokračuje po hranách určených písmeny ze zásobníku. Je-li dosaženo terminálního uzlu, nalezl se hratelný slovní tvar.

Vyhledávání v trie je tedy velmi rychlé, zvláště při neúspěšném hledání. Nicméně nevýhodou je značně velká spotřeba paměťového prostoru, neboť je ukládána informace pro velké množství stavů. Ačkoliv dochází ke kompresi společných prefixů, je potřeba prostorovou složitost dále snižovat. Zde hraje důležitou roli teorie jazyků a konečných automatů.

3.3 Konečné automaty

Před samotnými návrhy reprezentací vyhledávacích struktur založených na deterministických konečných automatech je zapotřebí stručně uvést základní definice a termíny.

3.3.1 Základní pojmy

Základním stavebním prvkem teorie jazyků je pojem *abeceda*. Tou se chápe libovolná konečná množina Σ a prvky této množiny se pak označují jako *znaky* (nebo také *symboly* či *písmena*). *Slovo* nad touto abecedou Σ je jakákoliv posloupnost symbolů z této abecedy. *Prázdné slovo*, značené jako ϵ odpovídá slovu nulové délky.

Množinu všech slov nad abecedou Σ značíme Σ^*, množinu všech neprázdných slov Σ^+. Nyní k samotné definici deterministického konečného automatu.

Definice. *Deterministický konečný automat* (dále jen DKA) $\mathcal{M}$ se definuje jako pětice $(Q, \Sigma, \delta, q_0, F)$, kde význam jednotlivých složek je následující:

- Q je neprázdná konečná množina *stavů*.
- Σ je konečná množina vstupních symbolů nebo také *vstupní abeceda*.
- $\delta : Q \times \Sigma \rightarrow Q$ je parciální *přechodová funkce*.
- $q_0 \in Q$ je *počáteční stav*.
- $F \subseteq Q$ je množina *koncových stavů* (nebo též *terminálních*).

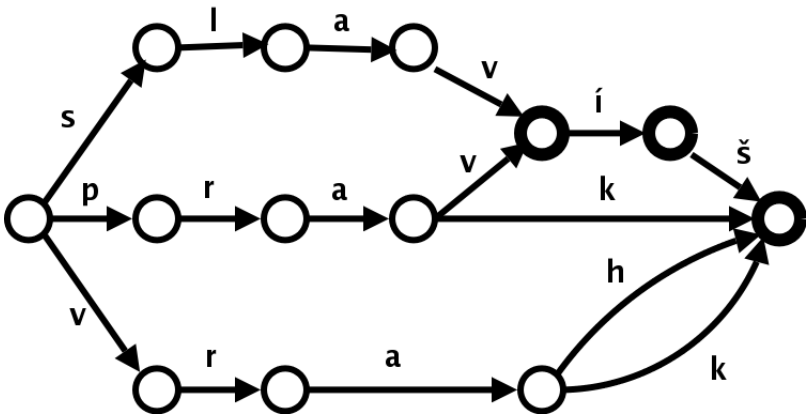
Pro formální popis jazyka rozpoznávaného definovaným deterministickým konečným automatem $\mathcal{M}$ je nutné zavést *rozšířenou* přechodovou funkci $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$, definovanou induktivně přes délku slova ze Σ^*:

- $\hat{\delta}(q, \epsilon) = q, \forall q \in Q$.
- $\hat{\delta}(q, wa) = \begin{cases} \delta(\hat{\delta}(q, w), a) & \text{je-li } \hat{\delta}(q, w) \text{ i } \delta(\hat{\delta}(q, w), a) \text{ definováno,} \\ \perp & \text{jinak.} \end{cases}$

Zápisem $\hat{\delta}(q, w) = s$ se rozumí, že automat $\mathcal{M}$ přejde pod slovem w ze stavu q do stavu s. Jazyk *akceptovaný* (*rozpoznávaný*) automatem $\mathcal{M}$, značený $L(\mathcal{M})$ je tvořen všemi slovy, pod kterými automat přejde z počátečního stavu q_0 do libovolného z koncových stavů. Formálně pak:

$$L(\mathcal{M}) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$$

Vztaženo ke Scrabble, jazyk $L(\mathcal{M})$ odpovídá právě lexikonu. V další části budou představeny některé metody konstrukce minimalizovaných, deterministických, acyklických konečných automatů (dále jen MDAKA), jejichž využitím se rapidně sníží výsledná prostorová složitost, zatímco bude zachována výhoda rychlého vyhledávání v těchto strukturách. Nutnou a postačující podmínkou pro to, aby DKA byl acyklický, je konečnost $|L(\mathcal{M})|$. Příklad takového automatu pro trie z obr. 3.1 ukazuje obrázek 3.2.



Obrázek 3.2: MDAKA

3.3.2 Techniky minimalizace a tvorby MDAKA

Tato část se zabývá konstrukcí *statických* MDAKA (tj. které je nutno vytvářet znovu při změnách lexikonu), neboť slovní zásoba pro Scrabble je víceméně stabilní a změny se většinou týkají odstraňování chyb a občasného přidávání

nových slovních tvarů. Budou představeny techniky, které jsou nejčastěji využívány při implementaci lexikonů deskových slovních her.

Obecně existují dva přístupy pro tvorbu MDAKA:

- **minimalizace trie** – probíhá ve dvou fázích, v první je zkonstruován samotný trie a nad ním následně provedena některá z minimalizačních metod, ať již generické pro konečné automaty [Wat95] nebo specializované pro acyklické KA [Knu86].
- **inkrementální konstrukce** – proces konstrukce probíhá v jednom kroku, automat je vytvářen postupným přidáváním slov a minimalizován za běhu. Těmto technikám se bude práce dále věnovat. Důležitým požadavkem ovšem bude, aby vstupní řetězce byly lexikograficky uspořádány, v opačném případě je získán nedeterministický konečný automat. Konstrukcí minimalizovaných deterministických konečných automatů z netříděných dat se zabývá například [Dac98].

3.3.3 Jednoduchý MDAKA

Jde o formu deterministického konečného automatu, ze které vychází dále prezentované metody konstrukce MDAKA. Na přechod se zde nahlíží jako na dvojici ⟨*znak, cílový_stav*⟩ a stavy jsou reprezentovány jako množiny svých *výstupních přechodů*.

Dva přechody jsou si *ekvivalentní*, jsou-li shodné jejich obě odpovídající složky. Podobně pro stavy platí, že dva stavy jsou ekvivalentní, pokud přesunutím *vstupních přechodů* jednoho stavu do druhého a následným odstraněním prvního stavu nedojde ke změně jazyka akceptovaného DKA. Přesněji, stavy s identickými množinami výstupních přechodů jsou si ekvivalentní. MDAKA tedy neobsahuje žádné 2 různé ekvivalentní stavy.

Následující algoritmus ilustruje konstrukci jednoduchého MDAKA:

```
01  s0 ='¶'; i = 0; larval_state[0] = ∅;
02  while not eof do
03      načti řetězec do s1[0 .. q − 1] a nastav q na délku s1;
04      s1[q] ='¶'; p = 0;
05      while s0[p] = s1[p] do p++; end while;
06      while i > p do
07          new_state = make_state(larval_state[i]);
08          i = i − 1;
09          larval_state[i] = larval_state[i] ∪ {⟨s0[i], new_state⟩};
10      end while;
```

```
11      while i ≤ q do
12          s_0[i] = s_1[i];
13          i = i + 1;
14          larval_state[i] = ∅;
15      end while;
16  end while;
17  while i > 0 do
18      new_state = make_state(larval_state[i]);
19      i = i − 1;
20      larval_state[i] = larval_state[i] ∪ {⟨s_0[i], new_state⟩};
21  end while;
22  start_state = make_state(larval_state[0]);
```

Algoritmus č. 1 – konstrukce jednoduchého MDAKA

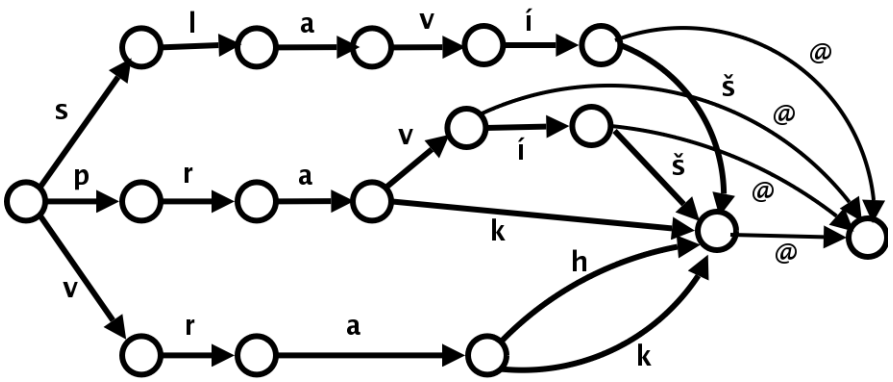
Tento algoritmus byl převzat z [CD01], kde je popsán podrobněji, idea algoritmu pochází z práce [Mih98] a je založená na přidávání stavů do konečnému automatu v takovém pořadí, aby nově vytvořený stav nemusel být již dále upravován a rozšiřován.

Na řádku **05** probíhá výpočet společného prefixu, cyklus na ř. **06** pak utvoří stavy pro sufix předchozího řetězce, cyklus na řádku **11** kopíruje sufix řetězce s_1 do s_0. Všechny přechody označené speciálním symbolem '¶' vedou do koncového stavu, jehož množina výstupních přechodů je prázdná. Výsledný minimalizovaný automat obsahuje právě jeden koncový stav. Pole $larval_state[]$ obsahuje množiny přechodů a slouží spolu s funkcí $make_state()$ k vyhledávání ekvivalentních stavů v dosud vytvořeného automatu. Není-li žádný takový nalezen, je příslušný stav přidán.

Na obrázku 3.3 je zobrazen příklad jednoduchého MDAKA pro trie z obrázku 3.1. Je vidět, že minimalizací se zredukoval počet stavů na téměř $1/3$ z původního počtu stavů v trie. Nicméně lze ještě dále tento počet snižovat vhodnou reprezentací stavů a přechodů. Například odstraněním přechodů označených symbolem '¶' (na obrázcích nahrazen symbolem '@'). Těmito úpravami se zabývají následující alternativy – Moorův a Mealyho stroj.

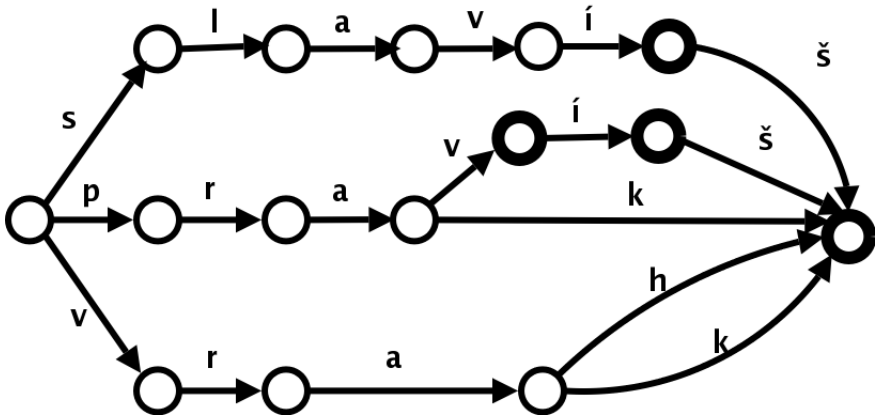
3.3.4 Moorův stroj

Moorovy stroje jsou konečné automaty, které nesou informaci o své struktuře „zakódovanou“ ve stavech. Stav je v tomto případě definován jako dvojice ⟨*množina_přechodů, terminální_příznak*⟩. Příznak určuje, zda je stav



Obrázek 3.3: Jednoduchý MDAKA

koncový či nikoliv. Na obrázku 3.4 se nachází ukázka Moorova stroje pro vzorový příklad trie (z obr. 3.1). Oproti jednoduchému MDAKA došlo k odstranění stavu se vstupními přechody označenými znakem '¶', které byly rovněž odstraněny.

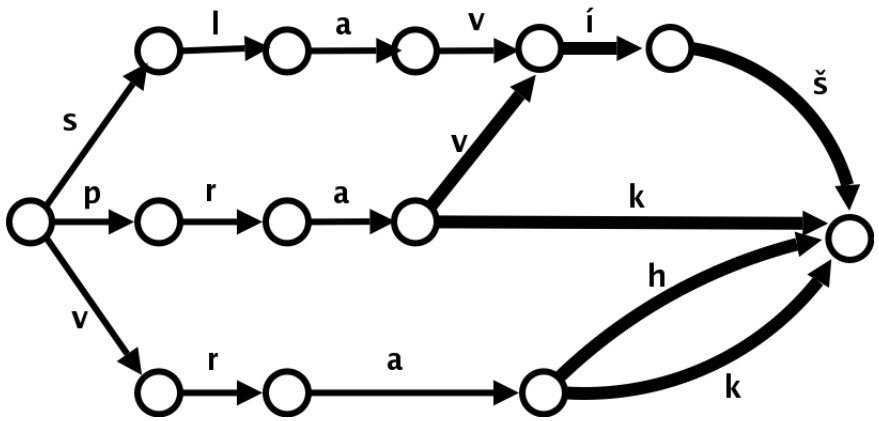


Obrázek 3.4: Moorův stroj

3.3.5 Mealyho stroj

Odlišný způsob vnitřní reprezentace MDAKA představují Mealyho stroje. Jejich stavba je zachycena v přechodech automatu. Přechod v Mealyho stroji je trojice ⟨ *znak, cílový_stav, terminální_příznak* ⟩, ekvivalence je definována

nad těmito trojicemi shodně jako u jednoduchého MDAKA. Stav je určen množinou přechodů. Jediným rozdílem v použití Mealyho stroje a předchozích dvou reprezentací je, že Mealyho stroj neumí akceptovat prázdné slovo. Obrázek 3.5 je ukázkou této reprezentace (tučně jsou zvýrazněny *koncové přechody*). Přímou konstrukci Moorova a Mealyho stroje ukazuje algoritmus



Obrázek 3.5: Mealyho stroj

níže, provedení pro obě reprezentace se liší v pouze označených místech.

01 **macro** $generic_make_state$(**var** i : **integer**);
02-MOORE $new_state = make_state(larval_state[i], is_terminal[i])$;
03-MOORE $i = i - 1$;
04-MOORE $larval_state[i] = larval_state[i] \cup \{\langle s_0[i], new_state\rangle\}$;
02-MEALY $new_state = make_state(larval_state[i])$;
03-MEALY $i = i - 1$;
04-MEALY $larval_state[i] = larval_state[i] \cup$
05-MEALY $\{\langle s_0[i], new_state, is_terminal[i+1]\rangle\}$;
06 **end macro**;
07 $s_0 =$'¶'; $i = 0$; $larval_state[0] = \emptyset$; $is_terminal[0] =$**false**;
08 **while not** eof **do**
09 načti řetězec do $s_1[0\,..\,q-1]$ a nastav q na délku s_1;
10 $s_1[q] =$'¶'; $p = 0$;
11 **while** $s_0[p] = s_1[p]$ **do** p++; **end while**;
12 **while** $i > p$ **do** $generic_make_state(i)$; **end while**;
13 **while** $i < q$ **do**
14 $s_0[i] = s_1[i]$;
15 $i = i + 1$;

16 $larval_state[i] = \emptyset$;
17 $is_terminal[i] =$**false**;
18 **end while**;
19 $s_0[q] =$'¶'; $is_terminal[q] =$ **true**;
20 **end while**;
21 **while** $i > 0$ **do** $generic_make_state(i)$; $i = i - 1$; **end while**;
22-MOORE $start_state = make_state(larval_state[0], is_terminal[0])$;
22-MEALY $start_state = make_state(larval_state[0])$;

Algoritmus č. 2 – konstrukce Moorova a Mealyho stroje

3.3.6 Porovnání

V této části jsou prezentovány výsledky, jakých lze dosáhnout aplikacemi předchozích tří algoritmů na různé typy lexikonů a vzájemné porovnání výsledných struktur. Výsledky jsou převzaty z [CD01] (kromě češtiny). Jedná se porovnání počtu stavů a přechodů každého z automatů a jejich výsledná velikost. Jako vstupní data byly vybrány soubory se slovní zásobou pro korektor překlepů, program *ispell*. (Pro češtinu byla použita vygenerovaná slovní zásoba implementovaného systému). Parametry vstupních souborů udává tabulka 3.1:

Lexikon	Velikost souboru [KB]	Počet řetězců	Průměrná délka	Velikost abecedy
Čeština	21 036	1 941 619	10.45	49
Angličtina	688	74 317	8.47	53
Francouzština	2 418	221 376	10.19	69
Němčina	2 661	219 862	11.40	58
Polština	16 447	1 365 467	11.48	64
Ruština	8 911	808 310	10.29	60
Španělština	7 187	642 014	10.46	56

Tabulka 3.1: Vlastnosti testovaných lexikonů

Tabulka 3.2 zachycuje porovnání počtu stavů výsledných konečných automatů zkonstruovaných zmíněnými technikami:

Lexikon	Jednoduchý	Moorův (term.)	Mealyho
Čeština	117 214	117 213 (17 435)	115 687
Angličtina	33 671	33 670 (5 192)	33 141
Francouzština	24 492	27 491 (3 610)	27 204
Němčina	46 436	46 435 (4 027)	46 050
Polština	67 486	67 485 (10 615)	66 268
Ruština	50 846	50 845 (8 742)	49 761
Španělština	34 778	34 777 (5 303)	34 231

Tabulka 3.2: Počty stavů 3 typů MDAKA

Vzájemné porovnání počtu přechodů obsahuje tabulka 3.3:

Lexikon	Jednoduchý	Moorův	Mealyho (term.)
Čeština	355 535	351 995	348 631 (98 066)
Angličtina	74 934	69 742	69 056 (14 693)
Francouzština	66 905	63 295	62 771 (14 095)
Němčina	90 737	86 710	86 266 (9 960)
Polština	199 397	188 782	185 952 (53 197)
Ruština	148 936	140 194	137 716 (47 625)
Španělština	111 345	106 042	105 170 (33 895)

Tabulka 3.3: Počty přechodů 3 typů MDAKA

Porovnání velikosti původního souboru s velikostí Mealyho stroje udává tabulka 3.4. Velikosti jsou udávané v kilobytech.

V literatuře zabývající se problematikou Scrabble jsou výše zmíněné struktury označovány spíše terminologií vycházející z teorie grafů a souhrnně jsou pak nazývány jako orientované acyklické grafy (OAG) [AJ88]. V naprosté většině implementací této deskové hry je lexikon ukládán ve formě OAG (nejčastěji Mealyho stroj, s informací kódovanou v přechodech automatu), a to díky značné úspoře paměťového prostoru a extrémně rychlému procházení struktur.

Jak už bylo zmíněno v kapitole 2.2 pojednávající stručně o pravidlech Scrabble, nová slova na desce mohou vznikat přidáním kamenů k již existujícím. Při generování tahů počítačem (viz kapitola 4) se hledá k již odehraným kamenům prefix a případně slovo rozšíří o vhodný sufix. V případě hledání sufixu je situace jednodušší, neboť průchod je prováděn „zleva“ ve směru

Lexikon	Vstupní soubor	Mealyho stroj
Čeština	21 036	1 361
Angličtina	688	270
Francouzština	2 418	245
Němčina	2 661	337
Polština	16 647	726
Ruština	8 911	538
Španělština	7 187	411

Tabulka 3.4: Porovnání velikostí

orientace hran, na rozdíl od generování prefixů, kdy jsou z písmen v zásobníku nedeterministicky utvářeny řetězce předem omezené délky, které jsou prefixem alespoň jednoho slova v lexikonu. Přestože je generování tahů velmi efektivní, snaží se i tento nedostatek odstranit následující struktura.

3.3.7 GADDAG

První zmínka o této struktuře se objevila v [AJ88] jako další možné urychlení generátoru tahů pro Scrabble. Mnohem důkladnější studii spolu s porovnáním nabízí [Gor94]. Princip spočívá ve zkonstruování speciálního „dvoucestného“ OAG (proto název GADDAG), který umožní snadné rozšiřování slov oběma směry, čímž eliminuje nedeterminismus při generování předpon (dále v kapitole 4). Každému znaku libovolného slova z lexikonu je přiřazena obousměrná cesta. Tím je umožněno pracovat při generování tahů s prefixy shodně jako se sufixy a je urychleno prohledávání odřezáváním neperspektivních cest.

Aplikací GADDAG je podle autora možné urychlit proces generování tahů až dvojnásobně (přesněji 2,19krát pro zásobník bez prázdného kamene a 2.66krát při držení obou prázdných kamenů) rychleji než při použití OAG. Tato optimalizace je však vykoupena až 5násobným zvýšením prostorové složitosti výsledné struktury.

Tabulka 3.5 je převzata z [Gor94] a ilustruje vzájemný poměr velikosti minimalizovaných OAG a GADDAG. Jako lexikon bylo použito druhé vydání oficiálního slovníku pro hráčskou komunitu Scrabble v USA z roku 1990, které obsahovalo 74 998 slov délky 2–8 znaků.

	OAG	GADDAG	poměr (G/O)
Stavy	17 856	89 031	4,99
Přechody	49 341	244 117	4,95
Velikost [B]	272 420	1 342 892	4,93
Bitů/znak	4,3	21,3	4.95

Tabulka 3.5: Porovnání počtu stavů a přechodů

Kapitola 4

Algoritmus pro generování tahů

Tato kapitola pojednává o nejdůležitější součásti všech implementací Scrabble, ve kterých může vystupovat jako protihráč počítač, tedy o hledání všech možných platných tahů pro dané rozestavění kamenů na hracím plánu a momentální obsah zásobníku písmen, zkráceně pak problém generování tahů. Prezentovaný algoritmus je de facto standardním algoritmem, který je využit v naprosté většině aplikací hrajících Scrabble, a to díky své extrémní rychlosti. Jeho autory jsou A. Appel a G. Jacobson, návrh algoritmu pochází již z roku 1988 [AJ88].

Výhodami tohoto algoritmu je redukování problému na jeden rozměr, omezeními kladenými při prohledávání, která umožní zmenšit prohledáváný prostor řešení s využitím backtrackingu, a vysokou efektivitou při tvorbě slovních tvarů pro jednotlivé tahy. Algoritmus tedy neprohledává celý lexikon, ale prochází všechna pole na hrací ploše, kde je možno umístit/připojit kameny k písmenů dříve odehraným (kromě prvního tahu samozřejmě). Poté se snaží inkrementálně utvářet platné slovo s využitím kamenů ze zásobníku a kamenů z desky. Přitom jsou udržovány struktury, které prohledávání učiní jednorozměrným, bez nutnosti opuštění jednoho řádku či sloupce hracího plánu.

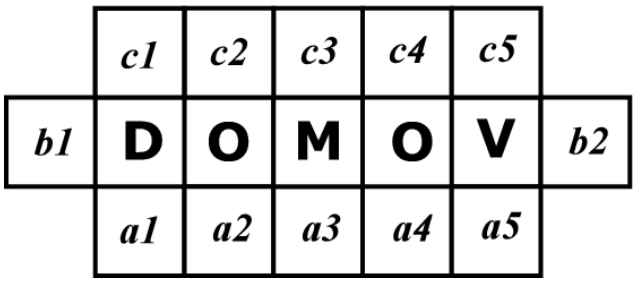
Jako vyhledávací struktura pro uložení lexikonu byl použit orientovaný acyklický graf (resp. Mealyho stroj, viz kapitola 3.3.5, str. 16), díky své nízké paměťové náročnosti, snadnému a rychlému průchodu. Nejprve však budou definovány a popsány základní pojmy a struktury, jež jsou algoritmem používány.

4.1 Redukce problému

Poněvadž pravidla Scrabble dovolují odehrát kameny pouze tak, aby byly všechny součástí jednoho řádku či sloupce (diagonální tahy povoleny nejsou), je možné tahy rozlišit na *vodorovné* a *kolmé*, a to podle směru tvorby slova. Bez újmy na obecnosti se lze omezit pouze na tahy vodorovné, neboť kolmé

tahy jsou v podstatě totéž co vodorovné, jen s hrací plochou transponovanou o 90° po směru hodinových ručiček. V dalším textu budou vodorovné a kolmé tahy jistým způsobem ztotožněny.

Definice. *Cross-množina*. Jde o průběžně aktualizovanou strukturu, která značně urychluje proces validace odehraných slov a zároveň pomáhá redukovat prohledávaný prostor při tvorbě slov. Jak již bylo zmíněno v pravidlech (2.2, str. 5), při vodorovném pokládání nových kamenů na desku musí všechna nová písmena tvořit platná slova i v kolmém směru (čtena shora dolů). Je tedy vhodné si pro každé prázdné pole na hracím plánu předpočítat množiny znaků, které jsou schopny se podílet na přípustných slovech v kolmém směru při vytváření slov vodorovných procházejících tímto polem. Tyto množiny se označují jako *cross-množiny*. *Triviální* cross-množina je cross-množina obsahující celou vstupní abecedu (dovoluje tedy na prázdné pole umístit jakýkoliv kámen). Situaci ilustruje obr. 4.1.



Obrázek 4.1: Výpočet cross-množin

Je odehráno slovo „domov“ a výpočet probíhá pro prázdná pole $b1$, $b2$, $a1$–$a5$ ve vodorovném směru. Podle umístění těchto polí je možné dále definovat *levou* (na obrázku pole $b1$, $c1$–$c5$) a *pravou* ($b2$, $a1$–$a5$) cross-množinu. V případě pravé varianty je situace jednodušší, nalezne se počátek kolmého slova a z kořene OAG provede průchod grafem po hranách odpovídajících písmenům tohoto slova. Cross-množinu následně určuje množina hran (resp. jejich písmenné ohodnocení) vycházející z uzlu grafu, kterého bylo dosaženo předešlým průchodem. Algoritmický zápis vypadá následovně:

01 $cross_mnozina(pole) = \emptyset$;
02 $pocatek_slova$ = NAJDI_NEJLEVEJSI($pole$);
03 $stav$ = TRAVERZUJ_GRAF($koren_grafu, pocatek_slova$);
04 **forall** h = DEJ_VYSTUPNI_HRANY($stav$) **do**
05 $\quad pismeno$ = DEJ_OHODNOCENI(h);

```
06      pom_stav = TRAVERZUJ_GRAF(stav, pismeno);
07      if JE_KONCOVY(pom_stav) then
08          cross_mnozina(pole) = cross_mnozina(pole) ∪ pismeno;
09      end if
10 end forall
```

Algoritmus č. 3 – výpočet pravé cross-množiny

V případě pole $b2$ z obrázku 4.1 získáme tímto postupem množinu {'*y*', '*ů*', '*a*', '*e*', '*u*'}.

Levá varianta (cross-množina pro $b1$) odpovídá generování jednopísmenných prefixů daného slova. Generování je nedeterministické a realizuje se následovně: z kořene grafu se prochází přes všechny hrany $h_i, i \in 1, 2..n$, kde n je počet výstupních hran kořene, do nějakého stavu S_j. Pokud je průchod ze stavu S_j po hranách odhodnocených písmeny slova z desky ukončen v koncovém stavu (odpovídající konci slova na hrací ploše), zařadí se písmeno hrany h_i do cross-množiny (viz algoritmus č. 4):

```
01 cross_mnozina(pole) = ∅;
02 pocatek_slova = pole + 1;
04 forall h = DEJ_VYSTUPNI_HRANY(koren_grafu) do
05      pismeno = DEJ_OHODNOCENI(h);
06      stav = TRAVERZUJ_GRAF(koren_grafu, pismeno);
06      pom_stav = TRAVERZUJ_GRAF(stav, pocatek_slova);
07      if JE_KONCOVY(pom_stav) then
08          cross_mnozina(pole) = cross_mnozina(pole) ∪ pismeno;
09      end if
10 end forall
```

Algoritmus č. 4 – výpočet levé cross-množiny

Pro pole $c2$ pravá cross-množina vypadá takto: {'*c*', '*d*', '*n*', '*t*'}. Cross-množiny je výhodné hledat před samotným započetím generování všech tahů a jejich aktualizace probíhá po každém odehrání kamenů, avšak počet prázdných polí, u kterých bude potřeba znovu spočíst cross-množiny je malý.

Definice. *Kotva.* Jedním z pravidel při pokládání nových kamenů na hrací plochu je, že alespoň jeden z nově položených kamenů musí stranově (horizontálně nebo vertikálně) sousedit s kamenem již odehraným (s výjimkou prvního tahu, kdy je deska prázdná). *Kotva slova* je definována jako nejlevější prázdné pole, které je přilehlé odehranému kameni. *Kotvy řádku* jsou

pak všechny kotvy slov nacházející se v jednom řádku. Z těchto míst je nejpřirozenější provádět hledání platných slovních tvarů. Na obrázku 4.2 jsou kotvami pole $a1$, $a2$ a $a3$.

			a1	P	Ř	E	D	E	a2	a3	S	T	O

Obrázek 4.2: Kotvy řádku

Problém nalezení všech hratelných slov v řádku je redukován na následující jednorozměrný problém: pro daný obsah zásobníku, daný obsah desky, spočtené cross-množiny a kotvy řádku, generuj přípustná slova v tomto řádku. Podmínka „nejlevější“ v definici kotvy slova zaručí, že každé slovo bude vygenerováno z kotvy právě jednou. Řešením specifikovaného problému se zabývá následující část.

4.2 Backtrackingový algoritmus

Úkolem zde předloženého algoritmu bude najít ve vodorovném směru z pozic určených kotvami všechny tahy. Kotva samotná je však součástí sufixu. Jelikož každé utvořené slovo může vzniknout složením prefixu (i prázdného) a sufixu, je proto i samotný algoritmus rozdělen do dvou částí:

1. Generování platných prefixů
2. Generování platných sufixů

4.2.1 Tvorba prefixů

Definice. *Platný prefix*. Platným prefixem (nebo také levá část) se rozumí kameny nalevo od kotvy. Levá část je pak tvořena buď kombinací písmen ze zásobníku (případ pole $a1$ na obr. 4.2), která je předponou nějakého slova z lexikonu, nebo písmeny již odehranými na desce (pole $a2$), avšak ne z obojího.

Na platnou předponu je kladeno i další omezení, a sice to, že každý prefix se musí rozkládat na posloupnosti hracích polí, která nejsou kotvami slova. Důvodem je zamezení duplikovanému vytváření týchž slov. Jako důkaz předchozího tvrzení nechť jsou dány kotvy na pozicích X a X+1 a generované slovo je „sto“ a hledá se pro obě pozice:

- pro kotvu na X: prefixem je ’s’ (nalevo od kotvy), sufixem pak „to“.

- pro kotvu na X+1: prefixem je „st“ a sufix ’o’.

Je-li pole nalevo kotvy prázdné, utváří se levý prefix (který může být i prázdný) z kamenů ze zásobníku. Pro každý takový prefix bude algoritmus předponu rozšiřovat o sufix počínaje kotvou, která je jeho součástí. Je-li naopak pole nalevo již obsazené, je platnou předponou posloupnost příslušných odehraných kamenů vlevo. Nyní bude diskutována první situace.

Snahou je omezit délku generovaného prefixu. Ta je omezena jednak počtem dostupných kamenů v zásobníku a dále také podmínkou z definice kotvy slova. Z tohoto důvodu nemůže levá část pokrýt kotvu, čímž je ohraničena maximální délka prefixu. Na obrázku 4.2 na straně 25 je vidět, že pro kotvu $a1$ je maximální možná délka předpony rovna 3, což se rovná počtu prázdných polí, která nejsou kotvami. Pro kotvy také platí, že jejich cross-množina je netriviální. Oproti tomu pole pokrytá levou předponu dovolují na jejich místo položit libovolné písmeno. Pro algoritmus generující levé prefixy je tedy potřebná znalost jejich max. délky. Samotná procedura s využitím techniky backtrackingu je zachycena v algoritmu č. 5:

```
01 function PREFIX(castecne_slovo, uzel N v OAG, limit){
02     SUFIX(castecne_slovo, N, kotva_slova);
03     if (limit > 0) {
04         for ( každou výstupní hranu E uzlu N){
05             if (písmeno p nad hranou E je v zásobníku) {
06                 odeber kámen k s písmenem p ze zásobníku;
07                 N' = uzel dosažený průchodem po hraně E;
09                 PREFIX (castecne_slovo + p, N', limit − 1);
10                 vrať kámen k zpět do zásobníku;
11             }
12         }
13     }
14 }
```

Algoritmus č. 5 – generování platného prefixu

V proceduře se vyskytuje volání SUFIX, které je definováno a blíže vysvětleno v následujícím textu. Parametr $limit$ je nalezená max. délka prefixu.

4.2.2 Tvorba sufixů

Narozdíl od prefixu, jenž může být prázdný, musí platný prefix obsahovat nejméně jeden znak a jeho utváření začíná na prázdném poli, které je kotvou (ta je z definice součástí přípony).

Generování sufixu (funkce SUFIX v algoritmu č. 6) je voláno pro každý dříve nalezený prefix (funkce PREFIX alg. č. 5) a druhým parametrem je uzel v OAG, kterého bylo dosaženo při hledání předpony. Další prohledávání je následně omezeno písmeny ze zásobníku , písmeny, která se nachází na desce (s využitím již odehraných kamenů tak mohou vznikat slova „obklopením“ existujících nově pokládanými) a cross-množinami prázdných polí.

Kdykoliv se v této fázi dosáhne v OAG koncového stavu, je příslušné slovo platným tahem (v algoritmu č. 6 funkce ZAŘAD_DO_SEZNAMU_TAHŮ). Tahy lze poté mnoha způsoby vyhodnocovat, popř. aplikovat některou ze strategií výběru nejoptimálnějšího tahu.

Následuje samotný algoritmus ve formě pseudokódu pro hledání sufixů:

```
01 function SUFIX(castecne_slovo, uzel N v OAG, pole){
02     if (je pole prázdné) {
03         if (N je terminální uzel) {
04             ZAŘAD_DO_SEZNAMU_TAHŮ (castecne_slovo);
05         }
06         for ( každou výstupní hranu E uzlu N){
07             if (písmeno p nad hranou E je v zásobníku) and
08                 p je prvek cross-množiny políčka pole) {
09                 odeber kámen k s písmenem p ze zásobníku;
10                 N' = uzel dosažený průchodem po hraně E;
11                 nasl_pole = políčko napravo od pole pole;
12                 SUFIX (castecne_slovo + p, N', nasl_pole);
13                 vrať kámen k zpět do zásobníku;
14             }
15         }
16     }
17     else {
18         p = písmeno z pole pole;
19         if (z N vede hrana ohodnocená p do nějakého uzlu N') {
20             nasl_pole = políčko napravo od pole pole;
21             SUFIX (castecne_slovo + p, N', nasl_pole);
22         }
23     }
24 }
```

Algoritmus č. 6 – generování platného sufixu

Stejně jako funkce PREFIX je i SUFIX rekurzivní funkcí, která vhodnými

testy prořezává prohledávací strom přípustných řešení (backtracking) a velice efektivní. V poslední části této kapitoly budou ještě zmíněny některé detaily výše prezentovaných algoritmů spolu s návrhem struktur, které mohou být využity při aplikaci herních strategií.

4.2.3 Detaily

Zde budou představeny některé situace, které ovlivňují chod diskutovaných algoritmů a navrhnuta struktura pro efektivní a snadný výpočet skóre jednotlivých tahů.

Prázdné prefixy. V případě, že pole nalevo kotvy slova (pro které je právě hledán tah) je obsazené odehraným kamenem, není utvářen žádný prefix a volá se ihned funkce SUFIX pro stav, jenž byl dosažen průchodem z kořene grafu po hranách ohodnocených písmeny obsazených polí.

Prázdné kameny (žolíci). Zvětšení prohledávaného prostoru nastává v situaci, kdy je v zásobníku umístěn žolík, který může reprezentovat libovolné písmeno. V algoritmech výše není případ výskytu žolíku explicitně testován, ošetření spočívá v přidání dodatečných testů ve funkcích PREFIX a SUFIX, přesněji do míst, kde je prohledáván zásobník písmen na přítomnost určitého znaku. Zde bude začleněn test, zda je k dispozici žolík. Jestliže ano, bude mu dočasně přiřazen znak, který je v zásobníku hledán. Není-li s pomocí prázdného kamene nalezeno platné slovo, je kámen vrácen zpět do zásobníku (řádek 13 v algoritmu 6) a opět nabude své polymorfní podoby.

Samozřejmě, že s nárůstem prostoru řešení se pochopitelně zvyšuje i počet nalezených tahů a podle toho se zvyšuje i časová složitost algoritmu, a to mnohonásobně. Největší nárůst je možné očekávat při výskytu obou (ve standardní distribuci písmen pro hru) prázdných kamenů v zásobníku. Nicméně v době vysoké výpočetní síly současných počítačů se výsledná časová náročnost v podstatě neprojeví, navíc počet prázdných polí je nízký a pravděpodobnost držení obou žolíku hráčem je celkem malá.

Struktura pro výpočet skóre. Jak již bylo zmíněno v části 4.1, základní ideou celého algoritmu pro generování tahů, je redukce problému založená na snížení dimenze prohledávaného prostoru. Nicméně výpočet skóre je stále prováděn nad celou hrací plochou, a proto je vhodné výpočet také jistým způsobem optimalizovat.
Optimalizaci je možno provést takto (pro obě orientace desky): pro každé neobsazené pole na desce spočíst celkovou sumu hodnot kamenů na polích, která se nachází bezprostředně od daného pole směrem na „sever“ a „jih“ (v tomtéž sloupci v řádcích nad a pod). Výpočet je vhodné umístit před

samotné vyvolání algoritmu pro hledání tahů, např. vhodným zařazením do procedury počítajících cross-množiny. Vyhodnocení skóre (včetně ošetření případných písmenných a slovních prémií) pak může proběhnout ve funkci ZAŘAD_DO_SEZNAMU_TAHŮ.

Kapitola 5

Herní strategie

Desková slovní hra Scrabble byla již ve svých počátcích navržena jako hra, která má za úkol prověřit hráčovu slovní zásobu a v jisté míře i jeho taktické schopnosti. Kromě toho ve Scrabble mnohé závisí na samozřejmě na štěstí a náhodě, zejména pak při tažení kamenů. Tato kapitola je zaměřena na ty faktory a nejrůznější herní situace, které mají dopad na průběh hry a mohou mít v konečné fázi vliv na výsledek odehrané partie, kdy je protihráčem počítač. Budou představeny heuristiky, které je možno aplikovat při výběru některého z nalezených tahů.

Avšak nutno předeslat důležitou vlastnost této hry, která značným způsobem znesnadňuje či dokonce znemožňuje uplatnění strategií, jakých bylo již v minulosti mnohokrát použito např. pro šachy, dámu, nebo u jiných strategických deskových her. Scrabble je totiž hra v literatuře ([Gor94], [AJ88]) označována z hlediska umělé inteligence jako *hra nedostatečné informace* (ang. „game of imperfect information“). Tato „nedostatečnost“ je zapřičiněna přítomností sáčku s písmeny, ze kterého se písmena v průběhu hry losují náhodně, a tak tedy nelze dedukovat obsah přihrádky protihráče ani množství a soubor písmen, která budou vytažena v následujících tazích. Oproti šachu, kde je herní situace i aktuální stav pro každého z hráčů jednoznačně určen rozmístěním figur na šachovnici, jsou tedy možnosti aplikovatelnosti taktiky podstatně ochuzeny.

Nicméně i přes uvedená omezení byly navrženy některé alternativy, které si za cíl kladou zavést v systémech implementujících deskovou hru Scrabble jistou formu umělé inteligence a taktického myšlení. Rozborem možností se zabývají následující podkapitoly.

5.1 Slovní zásoba

Modelování strategie by samo o sobě nepřineslo žádný efekt, pokud není k dispozici dostatečně robustní slovní zásoba. Dalo by se říci a je vcelku pochopitelné, že se úspěšnost programu v partiích vedených s lidským pro-

tějškem s rostoucím objemem lexikonu zvyšuje. Omezená slovní zásoba člověka nemůže v žádném případě konkurovat té, která je uchována v lexikonu (až stovky tisíc slovních tvarů). Už jen tento fakt je zárukou toho, že průměrný hráč Scrabble nemůže soupeřit s programem znajícím takové množství slov. V případě takovýchto hráču v podstatě nemusí být využity žádné speciální taktiky.

Opačná situace však nastává u vyspělých hráčů, expertů. Ti jsou jednak vybaveni zpravidla lepšími jazykovými dovednostmi, znají množství až obskurních slov (za všechny například slovo *xu*) a dokáží takticky myslet. Z tohotu důvodu vznikaly již od začátku návrhy strategií, které by alespoň zčásti mohly konkurovat i těmto expertním hráčům. Samozřejmě, že doposud zůstavá lidský mozek de facto „nepřemožen" (viz partie šachových velmistrů proti superpočítačům), a tak i taktika modelovaná strojem kýžené dokonalosti zatím nedosahuje, i když se stále častěji přibližuje.

5.2 Strategie výběru tahu

Problém stanovení tahu, jenž bude následně odehrán počítačem pro aktuální rozmístění hracích kamenů na ploše a daný obsah zásobníku písmen, je v podstatě hlavním úkolem nejrůznějších heuristik.

Standardně je nejdříve algoritmicky vygenerována množina všech přípustných tahů (např. algoritmem z kapitoly 4), které mohou bezprostředně následovat a nad touto množinou prováděny heuristiky, jež jako výsledek zvolí pro danou chvíli nejoptimálnější tah, resp. takový, jehož ohodnocení je v dané chvíli maximální. Popřípadě je zvolen tah, jehož vybrání může být součástí sledu tahů vedoucích k případnému vítězství. Základním problémem je tedy definice *heuristické evaluační funkce*, která každému nalezenému tahu přiřadí jeho „prospěšnost" pro další hru, ať již ve formě bodového ohodnocení či výhodných pozic na hracím plánu (bonusová pole). Nejčastěji aplikovanou strategií výběru tahu většiny programů implementujících Scrabble je hladová strategie.

5.2.1 Hladová strategie

Hladová strategie volí pro následující hru tah, jehož bodové ohodnocení je v dané situaci maximální. Na první pohled se volba tahu s nejvyšším skóre ve většině partií s průměrně zdatnými protihráči jeví jako dostačující, v případě expertních hráčů je již ne tak efektivní. Navíc, hladová strategie nemusí vést nutně k vítěznému zakončení hry a z tohoto důvodu byly vyvinuty některé další techniky výběru tahu.

5.2.2 Vyhledávací stromy

V úvodu této kapitoly bylo řečeno, že je Scrabble hrou s nedostatečnou informací k modelování některých strategických postupů. Jedním z nich byl myšlen tzv. *vyhledávací strom*, na kterém jsou většinou založeny heuristiky uplatňované u systémů implementujících jiné strategické deskové her jako jsou např. šachy nebo dáma. U těchto her je vždy přesně znám stav, ve kterém se nachází protihráč, spolu s možnostmi dalších tahů, které mohou v průběhu hry nastat (veškerá informace popisující daný stav je k dispozici).

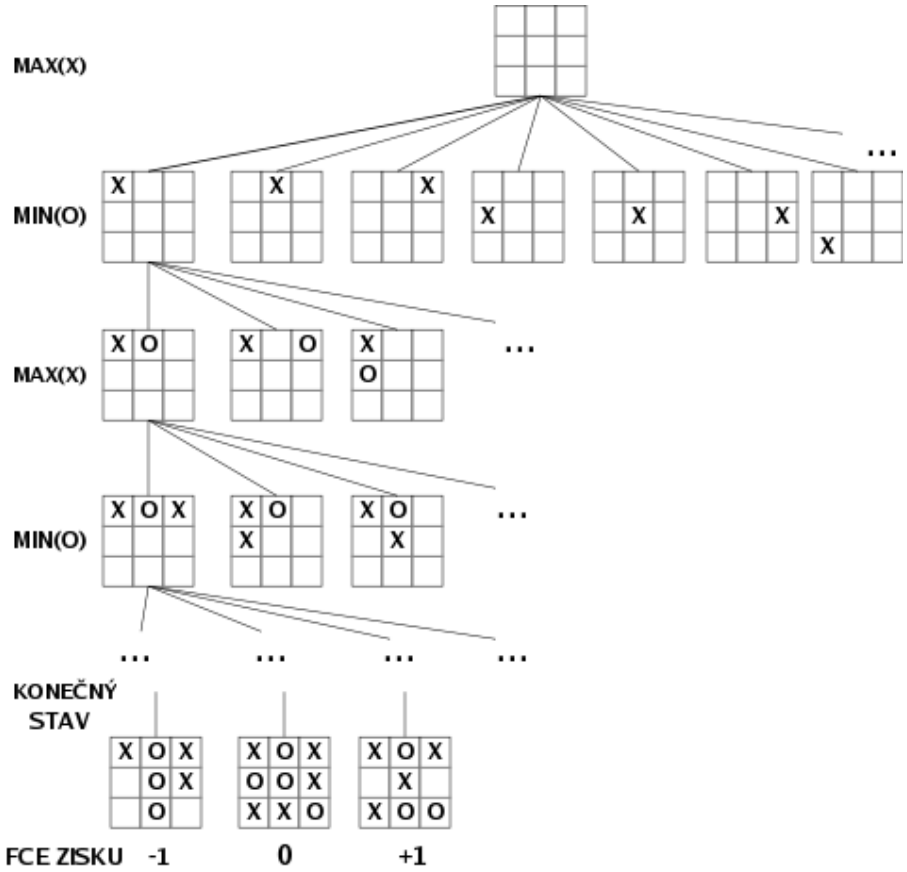
Na tomto místě je vhodné si říci, co vlastně pojem vyhledávacího stromu znamená. Jako nejvhodnější je použít terminologii převzatou z teorie umělé inteligence. V ní jsou hry chápány jako *vyhledávací problémy*. Jako *agenti* zde vystupují jednotliví hráči a bude uvažována hra pro 2 osoby s dokonalým rozhodováním. Tedy obecný případ hry se dvěmi hráči-protivníky, v literatuře označovanými jako MAX a MIN (MAX hru začíná). Hráči se pravidelně střídají a na konci hry je vítěznému hráči přiznána „odměna" (např. v bodech). Hru lze formálně definovat jako prohledávací problém. Složkami tohoto problému jsou:

- **počáteční stav** zahrnující nějakou pozici na hracím plánu spolu s informací, kdo je na tahu.
- soubor **operátorů**, které definují platné tahy, jež může hráč provést.
- **cílový test**, který zjistí, jestli je hra skončena či nikoliv, **konečnými stavy** jsou chápány ty, ve kterých hra končí.
- **funkce zisku** poskytující numerickou hodnotu popisující výsledek hry – např. vítězství, remíza nebo prohra (+1, 0, -1), popř. výsledný bodový zisk.

Instance uvedeného typu dat poté slouží jako vstup vyhledávacího algoritmu. Výstupem algoritmu je *řešení*, na které lze nahlížet jako na cestu z počátečního stavu do stavu, který splňuje podmínky cílového testu. Cílem hry MAXe je najít *strategii* vedoucí do konečného stavu bez ohledu na hru MINa.

Jako ilustrační případ vyhledávacího stromu je velmi často uváděn strom pro hru Piškvorky, pro jednoduchost se uvažuje varianta o 9 polích. Cílem je umístit 3 symboly do jedné řady, sloupce nebo diagonálně. Hru začíná MAX (symbol '**X**'). Podoba stromu je zachycena na 5.1.

Hra je odstartována s prázdnou mřížkou a MAX umísťuje symbol **X**, MIN pak **O**. Na obrázku 5.1 jsou pak vidět možnosti jednotlivých tahů



Obrázek 5.1: Vyhledávací strom

obou hráčů. Numerická hodnota v listech udává výsledek hry z pohledu MAXe.

U Scrabble je díky náhodnostním prvkům obsažených ve hře (především náhodný výběr kamenů ze sáčku) aplikování standardního vyhledávacího stromu vygenerovaného pro celou hru v současných podmínkách takřka neproveditelné, a to z důvodu vysokého faktoru větvení výsledného stromu. Je to dáno enormním množstvím možných kombinací písmen v zásobníku, dále odehratelných tahů, spolu s kombinacemi tažených písmen ze sáčku.

Nicméně existuje jistá situace v průběhu hry (resp. na samotném konci hry), kdy je možné případně kompletní prohledávací strom vývoje hry vytvořit. Jedná se o případ, kdy sáček s kameny již žádné další neobsahuje a dolosovat další již nelze. Následně odehrávaná slova tedy vznikají pouze

z těch písmen, jež jsou právě dostupná v zásobníku. V tomto speciálním stavu lze vydedukovat (v závislosti na počtu hráčů účastnících se partie) obsah zásobníků protihráčů, a to z kamenů odehraných na hracím plánu. Hra již obsahuje dostatečné množství informace a lze využít prohledávacího stromu k nalezení vítězné strategie v této koncové fázi hry.

Zde se uplatňují další techniky, které spočívají v „prořezávání“ stromu, s cílem vyřadit některé jeho větve (co možná nejvíc). Nejčastěji je pro tento případ využívána metoda *alfa-beta prořezávání*, jejíž přesnou definici a vlastnostni lze nalézt v literatuře zabývající se pojmy umělé inteligence. Přesto je tuto techniku pro Scrabble obtížné uplatnit, neboť faktor větvení činí zhruba 200 pro koncovou fázi hry se 7 kameny v zásobnících obou hráčů [She02] (pro porovnání – průměrný faktor větvení se např. u šachu pohybuje okolo 35).

5.2.3 Heuristiky

Cílem navrhovaných heuristik je definování *vyhodnocovacích funkcí*, které umožní posuzovat „kvalitu“ vygenerovaných tahů, evaluovat jejich prospěšnost. Nejčastěji se pro tyto účely využívají vážící funkce, jež přiřazují jednotlivým tahům váhy a z nich jsou následně vybráni kandidáti pro odehrání. Hlavními dvěmi objekty zájmu heuristik je zásobník písmen a hrací deska, které nyní budou diskutovány.

Heuristiky nad zásobníkem. Snahou heuristik prováděných nad obsahem zásobníku je zamezit odehrávání písmen nebo jejich kombinací, které mohou v dalším průběhu hry sloužit k tvorbě slov s vysokým bodovým ziskem. Přesněji, vyhodnocují se obsahy po odehraném tahu, tedy zbylé kameny. Jednotlivé heuristiky se zabývají těmito situacemi:

- **uchování slovotvorných kamenů**. Některé množiny písmen je výhodné si šetřit do dalších tahů, kde mohou být následně použity ke tvorbě hodnotných slovních tvarů. Příkladem (pro češtinu) jsou například časté koncovky slov: *-ají, -at, -ejší* nebo naopak předpony (*-nej*). V angličtině pak koncovky typu *-ing, -ly, -ed* . . . . Vyhodnocení, zda je jistou kombinaci písmen výhodnější si ponechat, je zpravidla nalezeno analýzou lexikonu na písmenné vzory s častou frekvencí výskytu, popř. jsou součástí tahů s vyšším ziskem.

- **omezení zůstatku stejných kamenů**. Zde je kladen důraz na to, aby po odehrání nezůstaly, pokud možno, v zásobníku dvojice (trojice) týchž písmen, které mohou „bránit“ v nacházení slov.

- **poměr samohlásek a souhlásek**. Některé kombinace mohou být opět vhodné ke tvorbě slov. V [She02] je uvedeno ohodnocení pro každý pár hlásek, páru *souhláska-samohláska* byla většinou přiřazena kladná a v ostatních případech záporná hodnota. Tyto hodnoty určují výsledné ohodnocení.

Kromě těchto heuristik bývá ještě zmiňována jedna teorie (více v [She02]), která tvrdí, že je výhodnější při prvním tahu neodehrávat „větší“ množství písmen, a to z toho důvodu, že v případě, kdy protihráč nemá přístup k žádné slovní prémii, byvá protihráčův následný tah bodově „chudý“.

Heuristiky nad deskou. Na hracím plánu se vyskytují pozice, ze kterých lze utvářet slova s vysokým bodovým ziskem. Typicky jde o prémiová pole, jež výsledné skóre mohou několikanásobně zvýšit. Cílem heuristik v této situaci je evaluace všech míst na desce mající potenciál zisku vyššího skóre a tato strategická pole využít či naopak neumožnit soupeři provést přes tato políčka svůj tah.

S hrací deskou se váže i následující heuristika. Ta se zabývá možnostmi protihráče odehrát v dalším tahu *bingo*, tedy všechny své kameny v zásobníku a získat tak bonus 50 bodů. Hledají se vhodní kandidáti na tah, který znemožní protihráči bingo získat, většinou je tato snaha doprovázena snížením bodového zisku z aktuálního tahu, avšak zabrání oponentovi odehrát vysoké skóre.

5.3 Výměna kamenů a *PASS*.

V průběhu hry je samozřejmě možné, že pro dané rozmístění kameny na hracím plánu a stav zásobníku písmen nebude vygenerován žádný platný tah. Na řadu musí tedy přijít možnost výměny některých svých kamenů, popřípadě provést vzdání se tahu – *PASS*. Do těchto situací se hra dostává většinou v koncové fázi hry anebo při „neslovotvorném“ obsahu zásobníku. Vzdání se tahu je poslední alternativou, po všech neúspěšných pokusech o bodový zisk (typicky v koncové fázi hry).

Co se týče výběru kamenů pro výměnu, jsou spočteny četnosti výskytu jednotlivých písmen v lexikonu popř. se frekvence počítají z odehraných slov většího množství partií, pokud jsou k dispozici. Písmena s nízkou četností jsou vybrány jako kandidáti na výměnu. Pro češtinu byla zvolena tato množina písmen: **Q, Ó, X, Ú, Ď, Ť, G, Ň, F**. V angličtině tuto množinu tvoří písmena jako **F, J, K, Q, V, W, X, Z**. Při výměně se prochází zásobník písmen a hledá některý ze znaků (v tomto pořadí) a kámen vymění.

Kapitola 6

Implementace

Následující kapitoly shrnují obsah praktické části této práce. Nejprve budou popsány cíle a požadavky, které byly vytyčeny jak zadáním, tak v průběhu prací. Dále pak je zachycen postup při vytváření slovní zásoby a provedeno zhodnocení výsledků, kterých bylo dosaženo. Nejprve budou zmíněny specifikace úkolů pro praktickou část.

6.1 Specifikace

Cílem praktické části bylo vytvořit program implementující deskovou hru Scrabble primárně určenou pro hru člověka s počítačem, v níž budou na straně programu začleněny jisté prvky strategického myšlení. Dalším úkolem bylo seznámit se s nástroji pro práci s českým korektorem pravopisu a využít je pro implementaci hry samotné. Následně pak zvolit některou strategii a zhodnotit chování a rozhodování programu v porovnání s reálnými partiemi, konkrétně byly k dispozici partie odehrané přes webové rozhraní hrající Scrabble. Elektronické partie z tohoto rozhraní pochází od Petra Kaňovského, odkaz lze nalézt na [CAS].

Následuje výčet dalších konkrétních požadavků na výsledný program, spolu s jejich specifikacemi:

1. Začlenění křížovkářského slovníku – v rámci Laboratoře zpracování přirozeného jazyka vznikl slovník vhodný k řešení a tvorbě křížovek. Úkolem bylo jej začlenit jako lexikon pro Scrabble. Více v části 6.3.

2. Významy slov – možnost získat význam některého slovního tvaru, který se vyskytuje v lexikonu. Pro tyto účely byl použit zmíněný křížovkářský slovník.

3. Možnost volby slovníku pro hru – externí lexikony.

4. Konfigurovatelnost – dovolit nastavení většiny prvků a pravidel hry, jako je definování sady kamenu a jejich ohodnocení, bodových hod-

not polí na hrací desce (včetně prémiových), velikost hrací plochy a zásobníku.

5. Práce se slovníkem – umožnit vytváření vlastních lexikonů pro hru, přidávání nových slov.
6. „Zakázání“ slovních tvarů – nepovolit použití některých slov, práce se seznamy těchto slov.
7. Umožnění uložit a načíst stav hry – klasická funkce herních programů.
8. Provádět logování – uchovávat průběhy her pro případné další zpracování. Vhodné pro nacházení statistických údajů či strategických postupů.
9. Uživatelské rozhraní – jednoduché ovládání, přehledný vzhled grafických prvků, spolu s nápovědou a dalšími užitečnými informacemi.

6.2 Použité programové vybavení

Aplikace vznikala jako jeden z projektů Laboratoře zpracování přirozeného jazyka (dále jen NLP lab.). Projekt samotný se nachází na stroji v adresáři `/nlp/projekty/pismenovka` na serveru `aurora.fi.muni.cz`. Jako cílový operační systém byl zvolen OS Linux, vývoj a testování pak konkrétně probíhal na platformách *i386* s OS Red Hat Linux 7.3, 8 a 9, spolu se strojem `aurora`. běžícím s OS Debian 3.0.

Jako programovací byl vybrán jazyk *C*, díky své rychlosti, která je základním požadavkem v prohledávací fázi hry. Navíc byl využit i pro spolupráci s morfologickým analyzátorem *ajka* ([Sed98]), dále programem pro vytváření kompaktních forem lexikonů a v poslední řadě proto, že zvolené prostředky pro vytváření GUI jsou nativně implementovány právě v tomto jazyce.

Uživatelské rozhraní bylo vyvíjeno pomocí knihoven **libgnome** a **libgnomeui** okenního systému **Gnome** verze 1.0 ve spojení s knihovnou pro snadnou tvorbu GUI – **GTK+** ve verzi 1.2. Jejich popis a referenční příručky lze nalézt v [GGAD] a [Ber02]. Jako poslední byly využity funkce knihovny **Glib**, taktéž verze 1.2., která nabízí prostředky pro efektivní a jednoduchou práci s množstvím datových a vyhledávacích struktur jako jsou lineárními seznamy nebo hašovací tabulky.

Program je distribuován a instalován klasickým způsobem pro OS Linux ve formě komprimovaných zdrojových textů za použití nástroje **GNU make** a je součástí přiloženého CD.

6.3 Vytvoření slovní zásoby

Jak již bylo zmíněno v kapitole 5 (str. 30), hlavním předpokladem k uplatnění účinných taktických postupů ve Scrabble je dostatečně mohutná slovní zásoba. Proto bylo potřeba tento aspekt zohlednit. Cílem bylo vytvořit lexikon s co možná nějvetším počtem slovních tvarů a zároveň byl kladen důraz na to, aby v maximální míře splňoval pravidla daná Českou asociací Scrabble ([CAS]) – pravidly jsou zde rozuměna omezení na přípustné slovní tvary (blíže v kapitole 2.2.6).

K dispozici byly dodány seznam hesel *Slovníku spisovného jazyka českého* (dále jen SSJČ) a oficiální seznam všech dvou- až pětipísmenných slov, která jsou povolena asociací Scrabble. Z těchto pramenů vznikal lexikon pro hru. Základním úkolem bylo vygenerování všech odvozených tvarů pro jednotlivá slova a následně provedeno označkování textu mluvnickými kategoriemi. Pro tento účel byla užita knihovna *alib*, která je součástí morfologického analyzátoru českého jazyka vyvíjeného v NLP lab. *ajka*. Autorem této knihovny a zároveň celého programu je Mgr. Radek Sedláček.

Z takto označkovaného textu byly nástroji pro zpracování textu a programovacím jazykem Perl postupně filtrovány jednotlivé slovní druhy. Pro každý slovní druh byly následně kladeny podmínky dané omezeními přípustností jednotlivých tvarů. Výsledkem je textový soubor o velikosti cca 20MB čítající 1 941 619 platných slovních tvarů délky 2–15 znaků. S příchodem nových, povětšinou přejatých slov, však bude potřeba lexikon průběžně aktualizovat.

Druhým slovníkem, který byl dodatečně přičleněn ke hře je slovník křížovkářský. Konkrétně byly k dispozici datové soubory s chemickými značkami, názvy měn, měst, podniků, letadel a mezinárodních poznávacích značek. Formát těchto souborů byl následující: na samostatném řádku je uveden řetězec `tvar@význam`, resp. naopak. V obou položkách se mohly navíc vyskytovat stylistické kategorie. Úkolem bylo formát sjednotit, odstranit chybné či chybějící údaje a provést výběr použitelných slovních tvarů. Po všech úpravách je k dispozici slovník se 41 887 slovy a jejich významy. Ty jsou taktéž využity v samotné aplikaci.

6.4 Realizace

V této části budou zmíněny implementační detaily týkající se realizace programu, přesněji formy uložení slovní zásoby, algoritmu pro generování tahů a implementované strategie. V další kapitole bude následně vyhodnoceno chování programu. Co se týče samotné realizace celého systému, celková

koncepce programu vycházela z poznatků z teoretické části práce v předchozích kapitolách. Snahou bylo „začlenit“ tyto znalosti do výsledné aplikace. Nakolik bylo dosaženo úspěchu v tomto směru přibližují následující oddíly textu.

6.4.1 Uložení lexikonu

Z možností ukládání lexikonu představených v kapitole 3 se jako nejvhodnější ukázal Mealyho stroj (resp. orientovaný acyklický graf), a to díky svým velice nízkým paměťovým nárokům (viz tab. 3.2 a 3.3) a velice efektivnímu způsobu průchodu touto strukturou. Pro vytvoření Mealyho stroje byl použit volně šířitelný kód, jehož autory jsou S. Deorowicz a M. Ciura (viz [CD01]). Důvodem k volbě byl jistě i fakt, že implementovaný algoritmus pro nalezení všech tahů podává nejlepších výsledků právě nad touto reprezentací.

6.4.2 Použitý algoritmus

Ačkoliv z porovnání algoritmů pro generování tahů nad OAG a GADDAG (kapitola 3.3.7) vyplývá, že hledání pro GADDAG probíhá zhruba 2krát rychleji než pro OAG, bylo přesto rozhodnuto implementovat algoritmus z kapitoly 4. Prezentovaný algoritmus je velice snadno pochopitelný a z tohoto důvodu i snadněji realizovatelný. Byl dokonce použit i v dosud nejlepší známé implementaci Scrabble, programu **Maven**, který je vyvíjen po dobu téměř 20 let [She02]. Navíc, nalezení všech slov tímto algoritmem pro danou situaci netrvá déle než 1.5s i v případě držení žolíka v zásobníku (na stroji AMD Duron 1000MHz, 256MB RAM), což je naprosto přijatelná doba na odezvu programu při hře (při držení obou trvalo samotné generování tahů okolo 3s).

6.4.3 Strategie

Začlenění strategického myšlení do vytvářeného programu bylo jedním z úkolů praktické části práce. V kapitole 5 byla diskutována většina používaných taktik, u některých byla zdůrazněna jejich neuplatnitelnost v reálných podmínkách, a to hlavně z důvodu jejich enormně vysoké časové složitosti. Z množiny prezentovaných postupů byly vybrány ty strategie, které bylo možné s danými znalostmi realizovat. Pro implementovaný systém byly zvoleny následující taktiky:

Bohatá slovní zásoba. Bylo již řečeno, že základním prvkem, jenž značně ovlivňuje úspěšnost počítačových oponentů ve hrách vedených proti lidskému protějšku, je mohutná slovní zásoba. Pro tento účel byl vybudován lexikon s téměř 2 miliony přípustných slovních tvarů, což je dostatečný základ pro uplatnění dalších strategií výběru optimálního tahu.

Hladová strategie. Strategie volby tahu s nejvyšším bodovým ohodnocením pro danou situaci je ve většině her zcela dostačující pro to, aby byl lidský protihráč poražen, proto je i tato strategie začleněna do programu. V kombinaci s předchozím bodem vytváří velice zdatného soupeře. Nicméně pro další zvýšení úspěšnosti ve hře vedené počítačem byla implementována další technika.

Heuristiky nad zásobníkem. Z heuristik uvedených v části 5.2.3 byla vybrána heuristika nad zásobníkem uvažující zůstatek shodných písmen po odehrání tahu, tj. pro zamezení držení kamenů, které by mohly ztížit tvorbu dalších slov v následujících tazích. V úvahu také připadá taktika uchování slovotvorných písmen, která má za následek vyrovnané bodové zisky v průběhu hry.

Poslední možností (pokud není bráno v potaz vzdání se tahu) je výměna nepotřebných kamenů. V případě „nouze“ (není-li např. nalezen žádný vhodný tah) je potřeba provést výměnu svých kamenů (je-li to možné). Byly spočteny četnosti výskytu jednotlivých písmen v odehraných partiích spolu s jejich zastoupením ve slovní zásobě. Tyto statistické údaje pak slouží k výběru „méněcenných“ kamenů pro výměnu.

6.5 Praktické výsledky

V tomto oddíle práce budou prezentovány a shrnuty výsledky, kterých bylo dosaženo v praktické části diplomové práce. V první fázi měl být implementován samotný systém hrající Scrabble, který by byl schopen porazit v podstatě většinu běžných hráčů.

Ačkoliv bylo nejvíce času v praktické části diplomové práce stráveno nad vývojem, laděním a testováním finální aplikace, nebudou zde popisovány jednotlivé funkce a možnosti aplikace, stejně tak jako technické detaily implementace systému jako použité datové struktury, programátorský popis algoritmů nebo ošetření chybových stavů. Efektivitu výsledného programu spolu s dalšími informacemi, které se týkají programovacích technik lze získat nahlédnutím do zdrojových kódů aplikace, které jsou k dispozici na přiloženém datovém CD. Ukázku grafické podoby programu lze nalézt v příloze B. Další text bude proto spíše zaměřen na vyhodnocení chování

programu a získání dalších statistických informací plynoucích z jeho činnosti.

Program samotný však není vybaven vyspělým strategickým myšlením, jakého je dosaženo například v [She02], neboť implementace je z tohoto hlediska značně časově náročná a navíc vyžaduje jistou znalost teorie umělé inteligence (herní prohledávací stromy). Avšak díky své bohaté slovní zásobě je i tak velice zdatným soupeřem.

Nicméně primárním úkolem bylo „nepřímo“ porovnat hru (resp. kvalitu) programu s hrou hráčů Scrabble. Z časových důvodů však nebylo možné nechat odehrát větší množství partií přímo s lidskými protějšky, a proto tato konfrontace probíhala pomocí jakési „simulace“ již odehraných her. K tomuto účelu bylo dodáno okolo 350 elektronických partií (elektronickou partií je rozuměn soubor se záznamem o průběhu hry – odehraná slova, stavy zásobníků, bodové ocenění tahů spolu s výsledným skóre). Konkrétně pak šlo o následující úkol s tímto zadáním: „Analyzovat úspěšnost programu v reálných hrách, pro dané partie určit, v jaké míře se program v dané herní situaci rozhodl lépe než jeho lidský protějšek“. Rozhodováním je myšlen výběr následujícího tahu a význam slova *lépe* je definován jako vyšší bodové ohodnocení nalezeného tahu, což je základním strategickým prvkem ve hře.

Před simulací samotnou bylo ještě nutné sjednotit formát všech souborů s elektronickými partiáři a tyto soubory následně připravit pro jejich postupné načtení programem pro simulaci výběru tahů. Díky oddělené implementaci grafické podoby a podpůrných algoritmů pro hru využívá simulační program shodných struktur a metod jako v případě samotné aplikace (procházení vyhledávací struktury, generování přípustných tahů, utváření cross-množin, výpočet skóre za tah, . . .).

Činnost simulačního programu lze charakterizovat takto – ze souborů s partiemi jsou postupně čteny informace o průběhu hry. Z těchto údajů je vytvořen jakýsi „vnitřní stav“ hry. Ten je definován aktuálním rozmístěním hracích kamenů (resp. písmeny na nich) na ploše a obsahem hráčova zásobníku s písmeny. Z takto nadefinovaného stavu je následně spuštěno „jádro“ simulace, které spočívá ve volání podpůrných algoritmů pro generování tahů a jejich ocenění. V další fázi je provedeno porovnání nalezeného, v dané chvíli „nejlepšího“ tahu, s tahem odehraným v analyzované partii. V každém průchodu jsou pak rozdíly v chování zaznamenány a závěrečnou fází simulace je statistické vyhodnocení získaných údajů, konkrétně pak o procentuální úspěšnost programu ve srovnání lidským oponentem.

Výstupem simulačního procesu jsou 2 typy souborů, lišící se pouze grafickou podobou. První je textový a ilustruje, jakým způsobem se v dané

situaci rozhodl člověk a jak generátor tahů. Ukázka výstupního souboru simulace je k dispozici v příloze A.3. V souboru jsou postupně zobrazeny obsahy zásobníků a souřadnice umístěných tvarů jak hráče, tak generátoru, v tabulce A.3 jsou vyobrazeny jen některé z těchto údajů. V zásobníku znak '$*$' (hvězdička) zastupuje prázdný kámen – žolíka. V odehraných slovech pak indikuje, že byl použit žolík pro následující písmeno. Uprostřed se pak nachází vlastní bodové ohodnocení provedených tahů. Jako poslední zde vystupují údaje o počtu nalezených tahů pro danou situaci a celkovou dobu v sekundách, kterou generátor pro tah spotřeboval. Z úsporných důvodů zde také nebudou zobrazeny průběžné stavy hrací desky. Ty je možné si prohlédnout v „grafičtější " podobě na přiloženém CD.

Celková doba simulace, kterou program prováděl nad 347 partiemi, byla 747.450 sekund (celkový čas strávený samotným generováním), v průměru tedy 2.154 partie za sekundu. Celkem bylo v této fázi vygenerováno 4 998 082 přípustných tvarů, pro situace se žolíkem 4 065 886, pro stavy bez držení žolíka 932 196. Ze schématu v tabulce A.3 na straně 50 je patrné, že nejvíce tvarů bývá nacházeno v situacích, kdy je v zásobníku přítomný žolík. Při vlastnictví obou prázdných kamenů je tedy možné získat až několik desítek tisíc vygenerovaných tahů.

Co se týče rozhodování programu v dané situaci je z tabulky A.3 vidět, že generátor v drtivé většině případů bodově předčil hráče. Velice často se také programu podařilo odehrát *bingo* s extra bodovým ziskem 50 bodů a odehrát tak všechna svá písmena ze zásobníku. Z výsledných časů je taktéž vidět, že implementace algoritmu pro generování všech tahů je velice efektivní.

Kromě použití elektronických partiářů k simulační fázi, byly provedeny i některé další statistické analýzy, které ilustrují naopak chování lidských hráčů. Kromě 347 partií obsahujících stavy zásobníků písmen bylo získáno i dalších 2700 odehraných her (už bez zásobníků), které byly podrobeny bližšímu zkoumání. Statistickým zpracováním byly získány údaje nacházející se v příloze A.1. První sada tabulek zde přibližuje četnosti nejčastěji utvářených slov ve hře, rozdělených podle počtu užitých písmen. Snadno je vidět, že nejčastěji tvořenými slovy jsou dvoupísmenné řetězce, s rostoucí délkou pak klesá i počet odehraných slov. Slova délky vyšší než maximální velikost zásobníku písmen pak samozřejmě vznikají přiložením nových kamenů k již existujícím. Navíc u slov kratší délky si lze všimnout snahy hráčů o získání co možná největšího bodového zisku použitím písmen s vysokým oceněním (písmena '**X**', '**G**', '**Ó**', . . .). Z úsporných důvodů je publikováno pouze prvních deset slovních tvarů pro každou délku. Údaje o kompletních statistikách jsou rovněž k dispozici na CD-ROMu, spolu s partiemi, které byly pro tento účel použity. V další části, v příloze A.2 je pak spočteno

zastoupení jednotlivých písmen abecedy ve slovech utvářených ve hře, setříděno podle absolutních četností. Z těchto údajů lze například zjistit, která písmena jsou „neslovotvorná“ a která se obtížně podílejí na vzniku nových platných slov. Tímto způsobem například vznikl seznam „nepotřebných“ písmen, která jsou vhodnými kandidáty na výměnu kamenů v situacích, kdy není k dispozici žádný hratelný tah.

Z dalších údajů jsou to pak průměrná délka odehrávaných slov a průměrná hodnota bodového zisku. Informace bylu získány z celkového množství 510 406 odehraných slov, průměrný počet dosažených bodů na jeden tah se pohyboval okolo 18.8447 (do této hodnoty nebyly započítány záporné hodnoty odpovídající odpisům ze skóre v závěru hry). Co se týče průměrné délky, činí tato hodnota 3.411.

Kapitola 7

Závěr a směry dalšího vývoje

Cílem předložené práce byla studie deskové slovní hry Scrabble. V úvodních kapitolách je tato hra stručně představena a pro úplnost byla v krátkosti vysvětlena základní pravidla této hry spolu s pojmy, které se v ní vyskytují.

Teoretická část práce pak přináší ucelený pohled na problematiku, jsou zde postupně diskutovány nejčastější způsoby ukládání slovní zásoby založené na různých technikách minimalizace konečných automatů, jsou popsány algoritmické postupy pro převod do této formy. Do fáze vytváření slovní zásoby byl začleněn morfologický analýzátor, byly využity jeho funkce pro lematizaci a generování derivátů. Navrhované struktury jsou dále podrobeny vzájemnému porovnání jak z hlediska efektivního průchodu těmito reprezentacemi, tak z hlediska paměťových nároků spojených s jejich tvorbou a následným uchováváním.

Další kapitola vyčerpávajícím způsobem pojednává o algoritmu generujícím platné tahy. V úvodu kapitoly jsou definovány fundamentální pojmy a součásti prezentovaného algoritmu. Je předložen backtrackingový algoritmus, jehož efektivita a rychlost spočívá ve výběru vhodných struktur a inkrementálnímu utváření přípustných slovních tvarů. Na závěr kapitoly jsou zmíněny některé detaily, které nejsou na první pohled patrné.

Možnostmi uplatnění strategického myšlení ve hře se zabývá kapitola 5. V ní je možné nalézt základní taktické postupy, které je možné ve hře využít. Zároveň jsou zde vysvětleny důvody, proč je naopak nemožné uplatňovat některé strategie běžné u jiných strategických her. V tomto aspektu vzniká prostor pro další případný výzkum. V současných podmínkách není zatím možné provádět generování kompletních herních prohledávacích stromů, které jsou de facto nejúčinnější metodou hledání chování programu přibližujícího se lidskému myšlení. Nicméně i bez nich existují systémy ([She02]), které s pomocí jiných, alternativních technik, dosahují excelentních výsledků i ve hrách s expertními hráči.

Cílem praktické části bylo vytvořit systém implementující samotnou hru s využitím poznatků z přechozích kapitol. Toto se dle mého názoru podařilo a vznikla tak plnohodnotná aplikace. Do fáze vytváření slovní zásoby

byl začleněn morfologický analyzátor, byly využity jím nabízené funkce. Přínosem programu je kromě radosti ze hry i možnost jejího využítí při hledání nových strategií popř. nacházení dalších znalostí (statistické údaje z logů) z odehraných partií. Spolu s informacemi o implementaci systému bylo rovněž provedeno zhodnocení chování programu ve skutečných hrách formou simulace na množině odehraných her a pro ilustraci provedeny některé statistické analýzy. Rozhodování programu je možné dále vylepšovat implementací dalších heuristik.

Literatura

[Gor94] Gordon, S. A. *A faster Scrabble move generation algorithm*, 1994, Software-Practice and Experience, vol. 24, p. 219–232.

[She02] Sheppard, B. *World-championship-caliber Scrabble*, 2002, Artificial Intelligence 134, p. 241–275.

[Bri59] Briandais, R. de la. *File searching using variable-length keys.*, 1959, Proceedings of the Western Joint Computer Conference, pages 295–298.

[CAS] Česká asociace Scrabble (ČAS). *Internetové stránky ČAS*, 2003, http://scrabble.hrejsi.cz/.

[Fred60] E. Fredkin. *Trie Memory.*, 1960, Communications of the ACM 3, 9(Sept. 1960), p. 490–500.

[GGAD] Pennington, H. *Gtk+/Gnome Application Development*, 2002, http://developer.gnome.org/doc/GGAD.

[AJ88] Appel, A. W., Jacobson, G. J. *The world's fastest Scrabble program*, 1988, Communication ACM, vol. 31, p. 572–578, 585.

[Dac98] Daciuk, J. *Incremental Construction of Finite-State Automata and Transducers, and their Use in the Natural Language Processing. Ph.D. thesis*, 1998, Politechnika Gdańska.

[Knu86] D. E. Knuth. *Computers & Typesetting, vol. C: TEX: The Program.*, 1986, Addison Wesley.

[Ber02] Beran, M. *Programování pro XWindow System*, 2002, http://www.ms.mff.cuni.cz/~beran/.

[Mih98] S. Mihov. *Direct construction of minimal acyclic finite state automata.*, 1998, Annuaire de l'Université de Sofia.

[Sed98] Sedláček, R. *Morfologický analyzátor češtiny*, 1998, Masarykova univerzita Brno, Fakulta informatiky.

[CD01] Ciura, M., Deorowicz, S. *How to squeeze a lexicon*, 2001, http://www-zo.iinf.polsl.gliwice.pl/~sdeor.

[Wat95] B. W. Watson. *Taxonomies and Toolkits of Regular Language Algorithms.*, 1995, Ph.D. thesis, Technische Unversiteit Eindhoven.

Příloha A

Statistiky

A.1 Statistiky odehraných slov

Délka (četnost)	2	3	4	5	6
Celkem tvarů	47 222	42 962	30 199	17 459	6 427
1.	ný (811)	ego (194)	sexy (61)	nahoď (14)	pověří (10)
2.	ex (803)	sex (190)	oceň (53)	doceň (13)	opaříš (7)
3.	go (802)	tex (182)	vaxy (49)	taxis (12)	ústavy (6)
4.	xu (750)	nýt (166)	sýci (48)	svěří (12)	svěříš (6)
5.	nó (744)	nús (157)	rexi (44)	ověří (12)	ošiďme (5)
6.	co (719)	sýc (147)	věci (43)	košať (11)	exnout (5)
7.	cé (714)	vax (144)	cosi (40)	hovět (11)	výšené (4)
8.	ký (663)	kýs (140)	únos (38)	výšen (10)	vyjíti (4)
9.	ch (656)	rex (134)	kýsi (38)	vnoří (10)	vústím (4)
10.	oř (652)	ceň (134)	zhať (37)	únosy (10)	vnořím (4)

Délka (četnost)	7	8	9
Celkem tvarů	3 104	1 913	310
1.	exnouti (4)	zatepáme (2)	nekřesali (2)
2.	vnoříme (3)	zahašený (2)	zvyšujete (1)
3.	uvěříme (3)	vyřešíme (2)	zvýrazním (1)
4.	utěšíme (3)	vštěpíme (2)	zvousatíš (1)
5.	utavené (3)	vrýpnout (2)	zvládnout (1)
6.	pověříš (3)	vinylový (2)	zůstavíte (1)
7.	pověřím (3)	uklidním (2)	zpucovati (1)
8.	ochoďme (3)	sledován (2)	zněmčilým (1)
9.	zvedáni (2)	schováme (2)	zkroužený (1)
10.	ztichnu (2)	potřáslo (2)	zježatění (1)

Délka (četnost)	10	11
Celkem tvarů	48	11
1.	zvládnouti (1)	uchrstnutém (1)
2.	zachrastíš (1)	podhazujeme (1)
3.	vyslaňovat (1)	nezvýrazním (1)
4.	vydupávati (1)	nezšedivělo (1)
5.	rozsekneme (1)	nezaspoříme (1)
6.	prošoustli (1)	nevyramuješ (1)
7.	prošetříme (1)	nerozvařila (1)
8.	postřihala (1)	nepřesíváme (1)
9.	podhazujem (1)	neprosvítáš (1)
10.	neztlumili (1)	neomalujeme (1)

A.2 Statistiky výskytu písmen ve slovech

Pořadí	Písmeno	Četnost	Pořadí	Písmeno	Četnost
1	o	34970	21	á	10304
2	e	31129	22	j	10270
3	a	28393	23	ř	10263
4	n	26040	24	š	10240
5	t	21210	25	é	10030
6	i	21018	26	z	9547
7	s	20012	27	ě	8413
8	v	18961	28	ý	8345
9	u	18646	29	č	6048
10	í	17726	30	ž	5512
11	m	17386	31	ď	5227
12	l	16077	32	ť	5051
13	d	15189	33	ó	5040
14	r	14915	34	f	4988
15	k	14671	35	x	4968
16	p	14476	36	ň	4750
17	h	14047	37	g	4540
18	c	12991	38	ú	4183
19	b	10955	39	ů	3099
20	y	10776	39	—	——

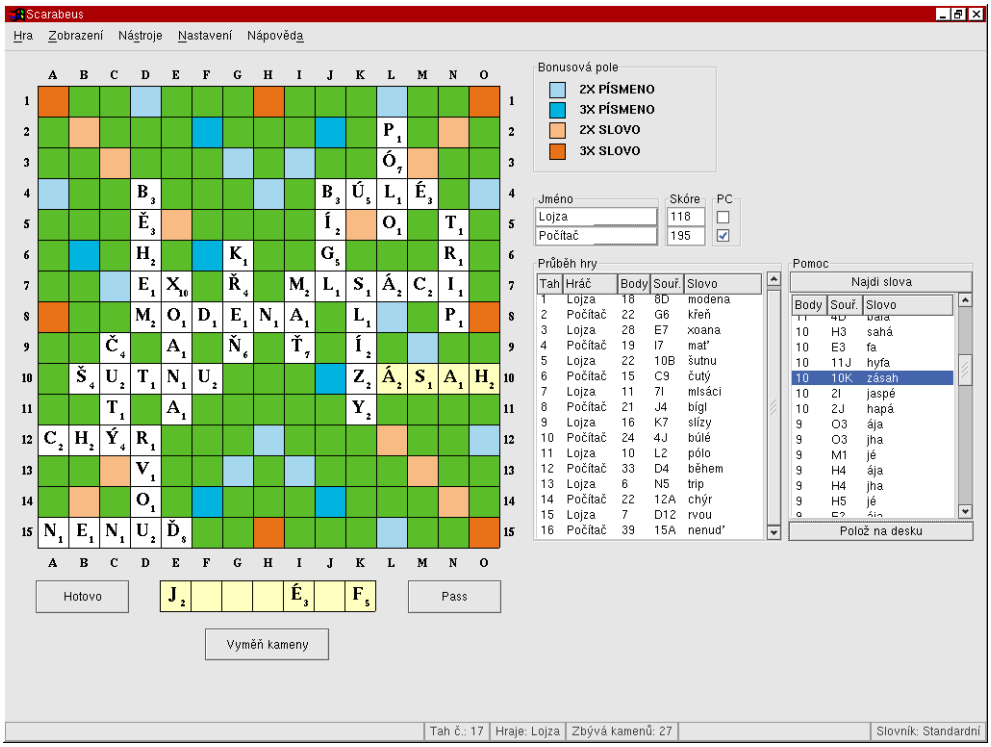
A.3 Ukázka výstupu simulace

HRÁČ			GENERÁTOR TAHŮ			
Zásobník	**Slovo**	**Body**	**Body**	**Slovo**	**Počet tvarů**	**Čas [s]**
pěal∗ru	pěla	12	74	u∗trpěla	3185	0,070
přrnýlp	planý	18	22	plný	58	0,010
∗ruřjmk	mřu	19	69	př∗ekrmuj	1503	0,140
řrpyxtb	by	18	24	brty	43	0,010
∗rjkitk	krti	10	21	kr∗ojit	1551	0,170
řrpxtgf	trp	12	17	grif	39	0,020
∗jkmenc	jme	18	22	n∗ájemci	2344	0,270
řxgfčod	oř	14	34	fox	75	0,020
∗kncolj	clo	15	24	col∗on	4557	0,340
xgfčdná	fám	19	25	fág	65	0,020
∗knjvoý	býk	16	37	výk∗lon	1815	0,390
xgčdnei	exni	32	32	exni	162	0,030
∗njvoid	dovij	21	82	n∗adojiv	3630	0,430
gčdtcvy	výt	11	46	vtyč	61	0,030
∗nievh∗	vine	14	83	∗li∗chevně	11507	1,750
gčdcyse	egy	34	34	egy	135	0,030
čdcsaůé	čas	20	20	čas	122	0,030
∗∗ueaon	neka∗š∗lou	87	88	neka∗nu∗lo	29407	2,600
...	...	...	...	...	...	...
ésásš	dáš	7	8	há	20	0,010
tšio	toť	18	18	toť	28	0,020
éss	pé	7	8	sdáš	7	0,010
ši	šil	6	6	šil	9	0,010

Tabulka A.1: Ukázka simulace partie

Příloha B

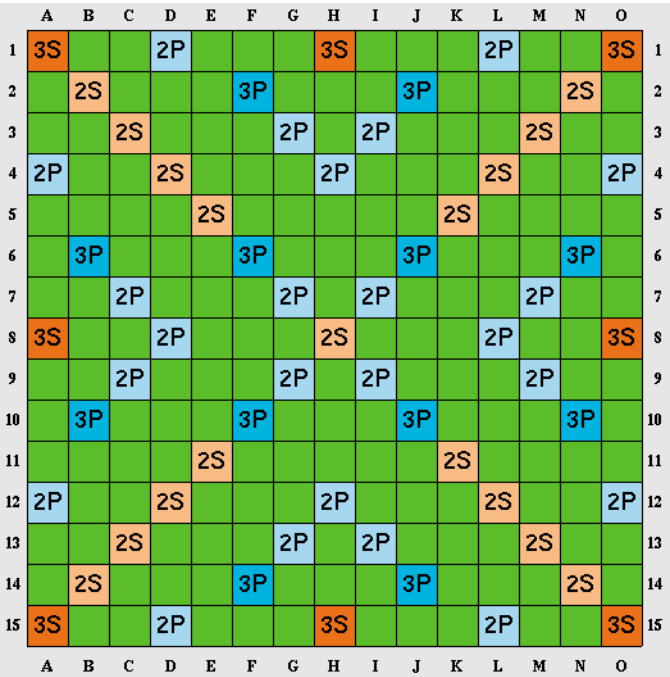
Ukázka grafické podoby programu



Obrázek B.1: Ukázka implementovaného systému

Příloha C

Podoba hracího plánu



Obrázek C.1: Hrací plocha