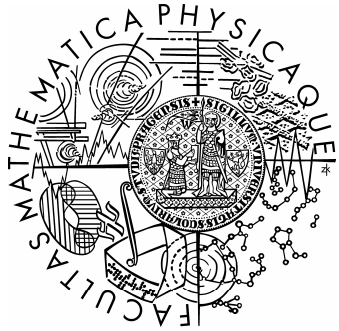
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Filip Dvořák

Scrabble

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Robert Babilon

Studijní program: Informatika, obecná informatika

2006

Poděkování

Děkuji Mgr. Robertu Babilonovi za poskytnutý konzultační čas a za poskytnutou volnost při realizaci aplikace implementující Scrabble.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Filip Dvořák

Obsah

Název práce: Scrabble
Autor: Filip Dvořák
Katedra: Katedra aplikované matematiky
Vedoucí bakalářské práce: Mgr. Robert Babilon
e-mail vedoucího: babilon@kam.mff.cuni.cz

Abstrakt: Tématem předkládané práce je studie deskové hry Scrabble a její převedení do elektronické podoby. Cílem práce je představení možných a používaných přístupů k implementaci systémů realizujících Scrabble a popsání a zhodnocení datových struktur a algoritmů v těchto systémech využívaných. Součástí práce je implementace zvolených řešení.

Po stručném seznámení se s pravidly Scrabble a základní analýze problému jsou v jednotlivých kapitolách teoretické části představeny datové struktury používané k uložení slovníku, algoritmy pro vyhledávání tahů a možné přístupy k herním strategiím. Praktická část práce pojednává o zvolených přístupech a zdůvodněních jejich volby. V závěru práce jsou shrnuty možné přínosy vzniklého systému a jeho další případná rozšíření.

Klíčová slova: Scrabble, GADDAG, heuristika, genetické algoritmy.

Title: Scrabble
Author: Filip Dvořák
Department: Department of Applied Mathematics
Supervisor: Mgr. Robert Babilon
Supervisor´s email address: babilon@kam.mff.cuni.cz

Abstract: The subject matter of the submitted thesis is a study of the board game Scrabble and its transfer to electronic form. The object of this thesis is to present possible and used approaches in implementation of Scrabble systems and to present and evaluate data structures and algorithms used in those systems. The implementation of the chosen approaches is a part of this thesis.

After the brief introduction into the rules of Scrabble and initiatory problem's analyse there are presented data structures used for dictionary storage, move generating algorithms and possible approaches in game strategies. The practical part of this thesis deals with chosen approaches and reasons why they were chosen. At the end of this thesis there is a summary of possible contributions of developed system and its possible following expansions.

Keywords: Scrabble, GADDAG, heuristic, genetic algorithms.

Kapitola 1

Úvod

Mezi deskovými hrami patří Scrabble v anglicky mluvícím zemích k jedněm z nejoblíbenějších her. Jeho popularita roste již po několik desetiletí a byl převeden do mnoha jazyků včetně češtiny. Oblíbenost hry je pravděpodobně dána jejími jednoduchými a srozumitelnými pravidly a lákavou kombinací využití jazykových znalostí, prvku náhody a strategického uvažování.

Teoretická část práce pojednává o použitelných přístupech k převedení hry do elektronické podoby. Pro uvedení do problému jsou nejdříve stručně popsána herní pravidla. V navazující kapitole budou představeny struktury použitelné k uložení slovníku. Následně bude podrobně popsán algoritmus hledání všech tahů. Poslední kapitola teoretické části se bude zabývat možnostmi uplatnění strategického uvažování ve hře.

V praktické části práce je nejdříve uvedena specifikace pro systém implementující Scrabble a následně jsou probrány formy realizace jednotlivých částí tohoto systému. V závěru jsou zhodnoceny míry dosažení zadaných cílů a možné přínosy vzniklého systému.

1.1 Úvod do problému

Nebudeme-li se zatím zabývat realizací uživatelského prostředí a praktickou implementací systému, zůstane nám jeden základní problém a ten zní: „Jak najít ten správný tah a co to vlastně je správný tah?“ To jsou sice otázky dvě, ale na tu druhou můžeme ihned odpovědět.

Správný tah je takový tah, který nám buď vyhraje hru nebo nám v příštím kole umožní najít další správný tah. Tato definice je příjemná, ale v případě Scrabble naráží na nedeterminismus vnášený náhodným výběrem hracích kamenů ze sáčku a neznalostí obsahu zásobníku protihráče, což je sice stále možné teoreticky modelovat konečným stavovým prostorem, ale mohutnost jeho větvení je neúnosně veliká pro jakýkoli způsob úplného prohledávání.

Správného tahu se tedy budeme muset vzdát a omezit se na tah výhodný, tedy tah, o kterém můžeme říci pouze to, že nám přinese nějakou výhodu, ať již bodovou, písmenkovou (zbavím se ošklivé kombinace písmenek) nebo třeba poziční (využiji dostupný bonus). Vidíme ale, že výhodnost je relativní pojem, a nemůžeme tedy nasadit stejnou laťku pro tahy přes všechna kola. Omezíme se proto na nalezení tahu z nějakého hlediska nejvýhodnějšího pro dané kolo, k čemuž nám stačí vyřešit otázku „Který ze

dvou tahů je nyní výhodnější?“, tedy konkrétněji najít relaci pro lineární uspořádání množiny všech možných tahů.

Tím se dostáváme k problému nalezení množiny všech možných tahů. Tento problém skrývá dva podproblémy. Prvním je zvolení vhodné datové struktury pro uložení slovníku všech slov povolených pravidly Scrabble a druhým, návazným problémem je vytvoření vhodného algoritmu pro efektivní hledání možných tahů pro danou herní situaci za pomoci slovníku.

Celkově jsme nalezli tři podproblémy: navržení vhodné datové struktury pro uložení slovníku, návazně vytvoření algoritmu pro vyhledání všech možných tahů a na závěr vytvoření vhodné relace „výhodnější“.

Kapitola 2

Stručně o Scrabble

V této kapitole bude krátce zmíněna historie a vývoj hry Scrabble ve světovém i tuzemském měřítku. Následně stručně popíšeme pravidla, a zároveň tak představíme termíny, které se budou vyskytovat v následujících kapitolách.

2.1 Historie

Koncepce hry, původně nazvané Lexico, vznikla v USA v roce 1938. Alfred Mosher Butts, v té době nezaměstnaný architekt, se rozhodl vytvořit novou deskovou hru, která by měla šanci najít si místo na trhu. Přišel s nápadem propojit slovníkové znalosti a prvek náhody. Na základě analýzy jazyka, ke které jako vstupní data posloužily noviny „The New York Times“, navrhl počty a hodnoty jednotlivých písmen. Toto původní rozvržení se v anglickém Scrabble zachovalo až dodnes.

Po proděličných začátcích došlo v roce 1952 k obratu, zapříčiněném zájmem prezidenta v té době největšího obchodního domu Macy´s, který se rozhodl přijmout Scrabble do své distribuce a prodávat ho ve velkém. Po několika přesunech nakonec skončila licence na hru Scrabble v rukou jednoho z největších světových výrobců hraček, společnosti Hasbro.

Autorem české verze hry Scrabble je Martin Hloušek. Česká verze Scrabble se poprvé objevila v roce 1993 a již v roce 1994 se konalo první mistrovství České Republiky. V roce 1998 následně vzniká Česká asociace Scrabble (ČAS) [1], která sdružuje kluby a hráče Scrabble, pořádá mistrovství a dohlíží na průběžnou aktualizaci slovníku.

2.2 Pravidla

Popisovaná pravidla jsou zkrácenou verzí pravidel českého Scrabble, která jsou v plném znění dostupná na internetových stránkách ČAS. Předně vysvětlíme pojmy, které se ve hře používají:

- Hrací kámen – představuje pevně danou dvojici písmeno a hodnota písmene. Zvláštním kamenem je žolík, který má nulovou hodnotu a není mu přiřazeno písmeno, dokud není umístěn na hrací desku.

- Hrací deska – čtvercová deska rozdělená na 15 × 15 čtverců, do kterých se při hře umisťují hrací kameny.

- Pole desky – jsou jednotlivé pozice k umisťování kamenů na hrací desce. Zvláštním polem je středové pole, kterým musí procházet první tah hry. Hrací pole mohou dále obsahovat bonusy, které mají význam při bodování tahu.

- Sáček s písmeny – obsahuje hrací kameny, které ještě nebyly přiděleny žádnému hráči.

- Zásobník hráče – představuje aktuální kameny daného hráče, ze kterých vytváří svoje tahy, a pokud ještě nějaké hrací kameny v sáčku s písmeny zbývají, pak svůj zásobník po odehraném tahu doplňuje. Každý hráč vidí pouze do svého zásobníku.

- Skóre hráče – všichni hráči začínají s nulovým skóre a to se během hry zvyšuje v závislosti na odehraných tazích. Po skončení hry se vítězem stává hráč s nejvyšším skóre.

2.2.1 Začátek hry

Prvním krokem je volba začínajícího hráče. Volba proběhne tak, že každý hráč si vytáhne ze sáčku jeden kámen a hráč s písmenem nejblíže začátku abecedy bude začínat. V případě shody se proces pro hráče, kteří měli stejné nejnižší písmeno, opakuje, dokud není právě jeden hráč s nejnižším písmenem. Kameny použité k tomuto výběru se opět vrací do sáčku.

Po zvolení začínajícího hráče si každý z hráčů náhodně vybere ze sáčku sedm kamenů a uloží si je do svého zásobníku tak, aby je ostatní hráči neviděli.

Začínající hráč následně vytvoří dle pravidel uvedených níže platný tah, tedy položí na hrací desku dva a více hracích kamenů a tento tah musí procházet středovým polem hrací desky.

2.2.2 Průběh hry

Hráč, který je na řadě, vybere ze svého zásobníku minimálně jeden kámen a položí jej na hrací desku. Následně si sečte body za všechna nově vzniklá slova a přičte si je ke svému dosavadnímu bodovému zisku. Svůj zásobník poté náhodně doplní kameny ze sáčku do počtu sedmi kamenů v zásobníku. V případě, že v sáčku není dostatek kamenů pro doplnění zásobníku, pak hráč dobere zbývající písmena ze sáčku.

Proti položenému tahu může být v okamžiku po jeho zahrání vznesen protest. Pokud je takový protest shledán oprávněným, tedy tah není v souladu s pravidly uvedenými níže, bere si hráč zpět položené kameny a ztrácí tah.

Namísto pokládání kamenů na desku může hráč svůj tah využít k výměně některých svých kamenů. Výměna probíhá tak, že hráč nejdříve zvolí kameny, které chce vyměnit, ze sáčku následně vylosuje stejný počet kamenů, které umístí do svého zásobníku, a původně zvolené kameny přesune do sáčku. Výměnu lze provést, pouze pokud v sáčku zbývá alespoň sedm kamenů.

Hráč se může svého tahu vzdát, a to tak, že oznámí ostatním hráčům PASS.

2.2.3 Pokládání kamenů

Alespoň jeden z nově položených kamenů musí stranově sousedit s některým kamenem, který už na desce ležel, výjimku tvoří první tah, který namísto sousedění musí procházet středovým polem.

Všechny položené kameny daného tahu musí ležet v jednom sloupci nebo jednom řádku a musí existovat slovo, na kterém se všechny podílejí.

Za vzniklé slovo považujeme na sebe navazující skupinu kamenů v řádku nebo ve sloupci, slova čteme shora dolů a zleva doprava. Každé nově vzniklé slovo, tedy slovo obsahující právě položený kámen, musí být přípustné.

Položí-li hráč žolíka, neboli prázdný kámen, musí říci, jaké písmeno žolík zastupuje, žolík pak dané písmeno zastupuje až do konce hry.

2.2.4 Bodování

Bodový zisk hráče za tah je daný součtem bodů všech písmen ve všech nově vzniklých slovech. Pokud je tedy například hráčem položený kámen součástí dvou slov, pak hráč získá jeho hodnotu dvakrát.

Hrací deska obsahuje zvláštní bonusová pole, která se týkají pouze počítání bodů. Bonusová pole jsou dvou druhů, bonus za celé slovo a bonus za písmeno. Když hráč položí kámen na pole s bonusem za celé slovo, pak si přičte namísto součtu všech písmen slova dvojnásobek až trojnásobek tohoto součtu. V případě položení kamene na pole s bonusem za písmeno dojde ke zvýšení hodnoty tohoto písmene ve všech součtech slov, kterých je částí, na dvojnásobek až trojnásobek. K uplatnění bonusu dochází pouze v okamžiku, kdy na něj byl právě položen kámen. Bonusy jsou na hrací desce vyznačeny následovně:

- Světle modrá pole zdvojnásobují hodnotu písmene.
- Tmavě modrá pole ztrojnásobují hodnotu písmene.
- Růžová pole zdvojnásobují hodnotu celého slova.
- Červená pole ztrojnásobují hodnotu celého slova.

Ve hře existuje další bonus nezávislý na hrací desce. Pokud hráč v jednom tahu položí všech sedm kamenů, pak získává jednorázový bonus 50 bodů k hodnotě tahu.

Na konci hry je skóre každého z hráčů sníženo o součet hodnot kamenů, které danému hráči zůstaly v zásobníku. V případě, že na konci hry jistý hráč nemá v zásobníku žádný kámen, pak je naopak k jeho skóre přičten součet hodnot všech nepoužitých kamenů ostatních hráčů.

2.2.5 Konec hry

Hra končí, pokud je sáček prázdný a některý hráč právě vyprázdnil zásobník. Hra také končí v případě, kdy ve dvou po sobě jdoucích kolech všichni hráči nahlásili PASS.

Kapitola 3

Datové struktury

V této kapitole představíme výhody a nevýhody jednotlivých přístupů k uložení slovníku co do paměťové náročnosti a efektivity navazujícího algoritmu pro hledání všech tahů.

Předně si určíme, jaké dotazy by měla datová struktura zodpovídat. Základním dotazem je, zda se jisté slovo nachází ve slovníku. Pro pozdější využití ve vyhledávacích algoritmech předpokládejme také dotaz na všechna slova, která lze vytvořit doplněním prefixu nebo sufixu k jistému slovnímu základu, kde písmena využitá pro prefix nebo sufix jsou do jisté míry omezena.

3.1 Setříděný seznam slov

Tento způsob uložení můžeme nazvat naivním. Je zde uveden pouze pro porovnání a zvýraznění výhod dalších struktur.

Ptáme-li se na výskyt konkrétního slova ve slovníku, musíme projít O(log n) slov, pro hledání metodou půlení intervalu. Při hledání sufixu pro daný slovní základ projdeme O(log n) slov + počet slov s daným prefixem. Problém nastává, hledáme-li prefix pro daný slovní základ. Pak musíme projít všechna slova ve slovníku. Řešením by bylo vytvoření další struktury reversně lexikograficky setříděného seznamu slov.

Celkově nám tedy bude vadit, že při vyhledávání načítáme slova, která nepotřebujeme, kterých je vždy alespoň O(log n), a při hledání prefixů a sufixů načítáme zbytečně celá slova, když nás zajímá jen část, kterou chceme doplnit.

Prostorová složitost je O(n).

3.2 Trie

V naivním přístupu jsme byli nuceni načítat vždy celé slovo, a tedy opakovaně načítat stejné posloupnosti znaků. Tento problém do jisté míry řeší použití trie.

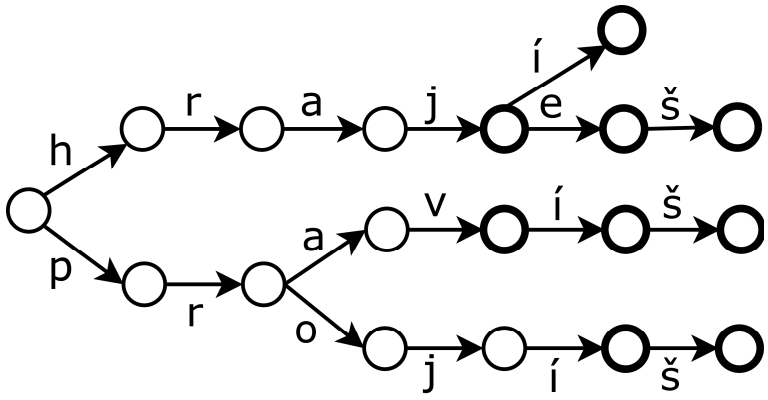
Vznik stromové struktury nazývané trie (název pochází z termínu „information retrieval“), je datován do roku 1959 [2]. Pro bližší pochopení této struktury definujme několik základních pojmů:

- X – množina všech slov ve slovníku.
- Σ – množina všech písmen vyskytujících se ve slovníku, neboli abeceda.

- Klíč – cesta z kořene do koncového uzlu, tedy posloupnost přechodů mezi uzly stromové struktury, představuje slovo ze slovníku.

Uložíme-li nyní do struktury trie náš slovník X, dojde k jistému ztotožnění stejných prefixů slov. Strom se pak bude v každém uzlu větvit nejvíce tolikrát, kolik je písmen v abecedě Σ. Vyhledání daného slova bude probíhat postupným průchodem strukturou trie od kořene ke koncovému uzlu a každý znak slova bude zpracován jen jednou při přechodu mezi danými uzly stromu. Klíč takového průchodu pak bude představovat hledané slovo.

Příklad trie pro množinu slov {hraj, hrají, hraje, hraješ, prav, praví, pravíš, projí, projíš} je na obrázku 3.1.



Obrázek 3.1: Trie

Použití trie přináší navíc určité přiblížení algoritmu pro vyhledávání k datové struktuře. Hledání všech možných sufixů pro daný slovní základ již nemusí probíhat odděleně pro každý sufix, ale využijeme stromovou strukturu trie, kterou projdeme pro daný slovní základ, a následně budou algoritmu předávány všechny sufixy získané průchodem do hloubky ze stavu daného slovním základem. Prakticky se samozřejmě omezíme na průchody, ke kterým lze použít písmenka hracích kamenů ze zásobníku hráče.

Rozeberme náročnost jednotlivých požadavků. Existenci slova ve slovníku ověříme v čase O(m), kde m je délka slova (což je v porovnání s naivní reprezentací prakticky čas O(1), neboli pouze jedno načtené slovo). Nalezení všech sufixů pro daný slovní základ a danou množinu písmenek je velmi rychlé, čím delší je slovní základ, tím řidší je podstrom a tím méně toho musíme projít. Nalezení všech prefixů pro daný slovní základ je náročnější, jsme nuceni procházet nejhustší část stromu, tedy od kořene dál, pro všechny výběry a jejich permutace ze zadaných písmenek. Protože jsme v nejhustší části

stromu, tak bude prohledávání selhávat pozdě a o to více jsme nuceni projít přechodů. K řešení posledního problému se dostaneme později.

Prostorová náročnost je silně závislá na slovníku. Pro slovník s mnoha sdílenými prefixy může být nižší nežli u lineárního seznamu slov, ale může být i výrazně vyšší, třeba v případě dlouhých slov bez společných prefixů.

Nabízí se otázka, jestli by ke snížení paměťové náročnosti nebylo možné využít i podobnosti sufixů. Za tímto účelem změníme lehce pohled na strukturu trie a budeme se na její stromovou strukturu dívat jako na acyklický deterministický konečný automat.

3.3 Konečné automaty

Bez hlubšího proniknutí do rozsáhlé formální teorie automatů a gramatik bude v tomto oddíle shrnuto využití konečných automatů v systémech realizujících Scrabble.

3.3.1 Základní pojmy

Deterministickým konečným automatem nazýváme pětici $(Q, \Sigma, \delta, q_0, F)$, kde význam jednotlivých složek je následující:

- Q – konečná neprázdná množina stavů,
- Σ – konečná neprázdná množina písmen, neboli abeceda,
- δ – zobrazení $Q \times \Sigma \to Q$, neboli přechodová funkce,
- $q_0 \in Q$ – počáteční stav,
- $F \subseteq Q$ – množina koncových stavů.

Dalšími pojmy jsou:

- λ – prázdné slovo,
- Σ^* – množina všech slov nad abecedou Σ, neboli všechny konečné posloupnosti písmen abecedy Σ.

Pro formální popis jazyka rozpoznávaného deterministickým konečným automatem je třeba zavést rozšířenou přechodovou funkci $\delta^* : Q \times \Sigma^* \to Q$, zavedeme ji induktivně:

- $\delta^*(q, \lambda) = q$
- $\delta^*(q, wx) = \delta(\delta^*(q,w),x)$

Jazyk L(A), rozpoznávaný deterministickým konečným automatem A, neboli slovník pro Scrabble, můžeme nyní formálně popsat následovně:

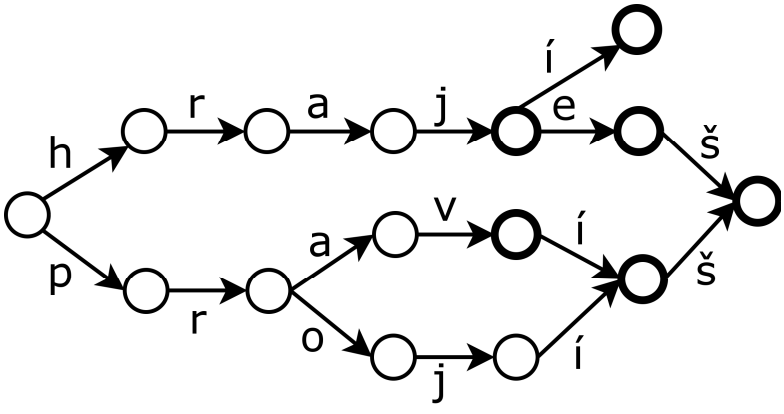
$$L(A) = \{w \in \Sigma^*, \delta^*(q_0, w) \in F)\}$$

3.3.2 Redukce konečného automatu

Pro konečné automaty vznikla řada algoritmů realizujících jejich redukci. Zaměříme se na algoritmy, které nepředpokládají přidávání nových slov do již vzniklého automatu, což je případ slovníku pro Scrabble, který se za běhu nemění. Tyto algoritmy bývají nazývány statické.

Statické algoritmy můžeme rozdělit do dvou kategorií. Algoritmy, které redukují již vzniklý konečný automat, a algoritmy redukující konečný automat během jeho konstrukce. Algoritmy z první kategorie jsou obvykle výpočetně náročnější, neboť jim předávané neredukované automaty mohou dosahovat značných rozměrů. Algoritmy redukující automat za běhu zpravidla vyžadují nějaké specifické podmínky pro svoji činnost, například acykličnost vznikajícího automatu nebo předtříděná vstupní data, což je pro některé aplikace omezující. Často se proto využívá kombinace algoritmů z obou kategorií. Automat je tak při konstrukci částečně zredukován a závěrečný algoritmus redukce pracuje s menším automatem.

Princip redukce konečného automatu je založený na sloučení ekvivalentních stavů automatu. Řečeno formálněji, vstupem redukčního algoritmu je konečný automat a jeho výstupem je konečný automat, jehož stavy jsou třídy ekvivalence nad stavy vstupního automatu. Pro acyklické deterministické konečné automaty bez nedostupných stavů, které nacházejí uplatnění v implementacích hry Scrabble, platí, že dva stavy jsou ekvivalentní, pokud se přesunutím všech vstupních přechodů z jednoho stavu do druhého a následným odstraněním prvního stavu nezmění jazyk rozpoznávaný automatem. Redukovaná podoba trie z obrázku 3.1 je znázorněna na obrázku 3.2.



Obrázek 3.2: Redukovaný automat

Následující algoritmus ilustruje konstrukci redukovaného automatu.

```
s0 ='¶'; i = 0; larval_state[0] = ∅;
while not eof do
  načti řetězec do s1[0 .. q − 1] a nastav q na délku s1;
  s1[q] ='¶'; p = 0;
  while s0[p] = s1[p] do p++; end while;
  while i > p do
    new_state = make_state(larval state[i]);
    i = i − 1;
    larval_state[i] = larval_state[i] ∪ { <s0[i], new_state> };
  end while;
  while i ≤ q do
    s0[i] = s1[i];
    i = i + 1;
    larval_state[i] = ∅;
  end while;
end while;
while i > 0 do
     new_state = make_state(larval_state[i]);
     i = i − 1;
     larval_state[i] = larval_state[i] ∪ { <s0[i], new_state> };
end while;
start_state = make_state(larval_state[0]);
```

Algoritmus 3.1: Konstrukce redukovaného konečného automatu

Algoritmus 3.1 byl převzat z [3], kde je také podrobněji popsán. Idea algoritmu je založena na přidávání stavů do konečného automatu v takovém pořadí, aby nově vytvořený stav již nemusel být dále upravován.

V literatuře zabývající se systémy realizujícími hru Scrabble bývá často nahlíženo na datové struktury pro uložení slovníku prostřednictvím teorie grafů. Pro redukovaný konečný automat je možné se setkat se zkratkou DAWG (Directed Acyclic Word Graph) [4].

3.4 Slovník cyklických transformací

Slovník cyklických transformací vytvoříme tak, že slovo nejdříve zrcadlově obrátíme a uložíme ho do nového slovníku. Na konec obráceného slova pak přidáme zvláštní znak, který se nevyskytuje nikde ve slovníku, a do nového slovníku zapíšeme všechna slova vzniklá postupným přesouváním prvního znaku slova na konec slova. Na příkladě slova „ráno“ mi vzniknou slova „onár“, „nár@o“, „ár@no“ a „r@áno“, kde za speciální znak jsme zvolili „@“.

Pokud takto upravený slovník převedeme do trie, pak se problém hledání prefixu pro daný slovní základ výrazně zjednoduší, neboť můžeme projít slovníkem nejdříve pro

slovní základ, tedy pozpátku (proto také slova zrcadlově obracíme), čímž se dostaneme do řidší části stromu a hledání uplatnění písmenek pro nás bude rychlejší.

K tomuto zlepšení by nám však stačilo vytvořit slovník obracených slov. Cyklické transformace nám dále umožňují přejít z prefixu na sufix v jakékoli části slova, a můžeme tak sjednotit operace hledání sufixu a hledání prefixu, a to tak efektivně, že slovní základ, na který se napojujeme, načítáme pouze jednou za celé prohledávání. Například pro slovní základ „moc“ projdeme nejdříve slovníkem pro „com“, následně budeme hledat všechny prefixy a pro každý nalezený, včetně prázdného, se přesuneme znakem „@“ do hledání sufixu. Nová slova tedy vyhledáváme ve tvaru *<obrácený slovní základ><obrácený prefix>@<sufix>*.

Tato struktura umožňuje z probraných variant implementace slovníku nejefektivnější vyhledávání nových tahů. Zvyšují se tím ale paměťové nároky, které mají tendenci se zvětšit více jak pětkrát, v závislosti na slovníku, oproti uložení běžného slovníku do trie.

3.5 GADDAG

GADDAG [5] je řešení používající slovník cyklických transformací a datovou strukturu rozšířené trie. Rozšíření je představováno přidáním další informace do uzlů. Každý uzel trie obsahuje ještě navíc informaci, jaký následující znak bude ukončovat slovo. Každé slovo uložené do trie je tedy zkráceno o poslední znak a ten je uložen do odpovídajícího uzlu. Tato úprava prakticky ruší pojem koncového uzlu.

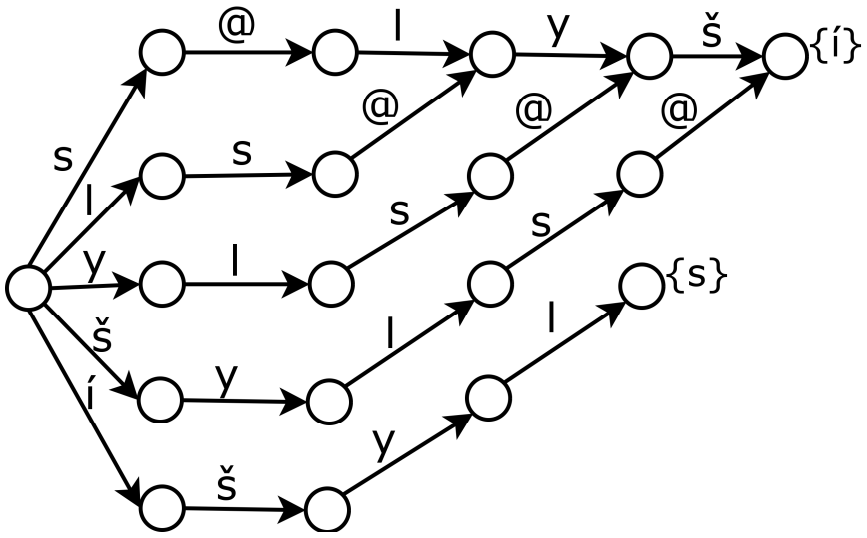
Toto rozšíření umožňuje sjednotit uzly, které by jinak možné sjednotit nebylo. Na příkladě slov „nízkost“ a „oblost“, nemůžeme při použití koncových stavů sjednotit pro tyto dvě slova sufix „ost“, protože slovo „nízko“ existuje, ale slovo „oblo“ nikoli. Uzel, na který by se přecházelo písmenem „o“, by pak neměl jednoznačně určeno, jestli je, nebo není koncový. Díky rozšiřujícím informacím v uzlu ale toto sjednocení můžeme provést, protože zakončení slova „nízko“ proběhne ještě v uzlu, na který ukazuje písmeno „k“ (v množině přiřazené danému uzlu bude nastaven bit odpovídající písmenu „o“) a uzel, na který ukazuje hrana pro písmeno „o“ je tak na existenci slova „nízko“ nezávislý.

Pro konstrukci GADDAG byl v [5] navržen algoritmus, který nechává automat vznikat v částečně redukované podobě, a snižuje tak vyšší paměťové nároky dané použitím slovníku cyklických transformací. Převzatý algoritmus 3.2 ilustruje konstrukci částečně redukovaného GADDAG. Příklad automatu zkonstruovaného tímto algoritmem pro slovo „slyší” je na obrázku 3.3.

```
forall slova ze slovníku tvaru a1,a2,...,an
  uzel = kořen_stromu;
  for i from n downto 3
    přidej_uzel(uzel, ai);
  přidej_závěrečný_uzel(uzel, a1, a2);

  uzel = kořen_stromu;
  for i from n-1 downto 1
    přidej_uzel(uzel, ai);
```

```
  přidej_závěrečný_uzel(uzel, @, a_n);

  for m from n-2 downto 1
    pom = uzel;
    uzel = kořen_stromu;
    for i from m downto 1
      přidej_uzel(uzel, a_i);
    přidej_uzel(uzel, @);
    přidej_hranu(uzel, a_m+1, pom);

přidej_uzel(uzel, znak) {
  if neexistuje_hrana(uzel, znak);
    pom = nový uzel;
    přidej_hranu(uzel, znak, pom);
  else
    pom = hrana(uzel, znak);
  uzel = pom;
}

přidej_závěrečný_uzel(uzel, znak1, znak2) {
  přidej_uzel(uzel, znak1);
  nastav_terminál(uzel, znak2);
}

přidej_hranu(uzel1, znak, uzel2) {
   // do uzlu1 přidá pro znak přechod na uzel2
}
```

Algoritmus 3.2: Konstrukce částečně redukovaného GADDAG



Obrázek 3.3: GADDAG pro slovo „slyší“

Kapitola 4

Vyhledání všech tahů

V roce 1988 vyšla publikace [4] popisující mimo jiné algoritmus pro hledání všech možných tahů. Tento algoritmus je od té doby s menšími úpravami používán v naprosté většině implementací hry Scrabble.

4.1 Základní myšlenka

Nebudeme procházet slovník a pro nalezené tvary hledat uplatnění na hrací ploše, ale místo toho budeme procházet hrací plochu a hledat návaznosti, kam bychom mohli připojit slovo. Tato metoda pak spolu se slovníkem uloženým do trie umožňuje značně snížit počet probíraných možností.

4.2 Redukce problému

Dle pravidel Scrabble můžeme každý platný tah označit buď za horizontální nebo vertikální, tedy tah buď nevybočuje z jednoho sloupce, nebo z jednoho řádku. Navíc můžeme uvažovat pouze tahy horizontální, neboť vertikální tahy jsou tahy horizontální s transponovanou hrací deskou.

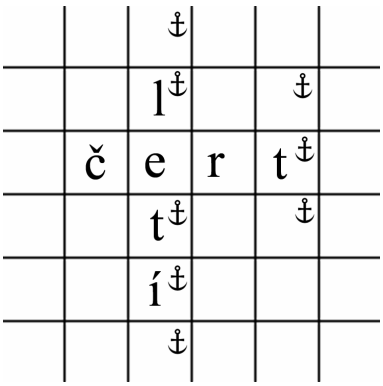
Hrajme tedy tah horizontální. Takový tah musí splňovat podmínku, že pokud se dotýká nějakého slova ve sloupci, pak v tomto sloupci také musí vytvořit nové platné slovo. Nové slovo se tedy od starého liší jen o jediné přidané písmeno z našeho tahu. Jaká písmena mohou rozšířit dané slovo, je možné již dopředu vyhledat, a protože jiná interakce s ostatními řádky neprobíhá, zredukovali jsme problém do jedné dimenze. Tímto předvýpočtem zjednodušíme práci algoritmu pro vyhledávání, který je vzhledem k velkému množství probíraných možností citlivější na jakékoli další výpočty, které by musel při hledání tahů provádět. Množině písmen, které mohou na dané pozici rozšířit slovo, budeme říkat cross-množina.

Nalezení cross-množiny je možné provést jednoduchým průchodem datovou strukturou slovníku pro položené slovo a prohledáním všech možností zakončení slova. Navíc pokud pracujeme se slovníkem cyklických transformací, pak je možné vyhledat obsah dvou cross-množin, resp. jednoho páru na začátku a konci slova, v jednom průchodu slovníkem. Stačí projít slovníkem pro dané obrácené slovo a množina jednopísmenných prefixů bude odpovídat množině koncových písmen v uzlu. Pro nalezení jednopísmen-

ných sufixů provedeme ve slovníku přechod pro znak „@“, který nás přenese do odpovídajícího uzlu.

Podíváme-li se následně na řádek hrací plochy, můžeme určit pozice, ze kterých bude výhodné spouštět prohledávání tak, aby žádný tah nebyl nalezen dvakrát a zároveň jsme byli nuceni projít co nejméně přechodů po hraně ve slovníku. Takovým pozicím budeme říkat kotvy.

Uvažujeme-li pouze tahy horizontální, což můžeme, neboť tahy vertikální získáme transpozicí matice, pak je možné jako kotvy pro horizontální tah zvolil pozice, kde leží buď neprázdná vertikální cross-množina, která není zároveň horizontální cross-množinou, nebo poslední písmeno horizontálně položeného řetězce písmen. V příkladu na obrázku 4.1 jsou značkem kotvy vyznačena všechna pole, která by byla vybrána pro započetí hledání nového horizontálního tahu.



Obrázek 4.1: Kotvy

4.3 Algoritmus

Samotný algoritmus můžeme následně rozložit na dvě části. Část hlavní, která pro všechny kotvy na řádku volá část vedlejší, a část vedlejší, která je představována dvojicí rekurzivních funkcí hledajících všechny dostupné tahy vázané na danou kotvu. Algoritmus 4.1, převzatý z [5], ilustruje průběh vyhledávání všech tahů pro zadanou kotvu. Algoritmus předpokládá, že slovník je uložený ve struktuře GADDAG.

```
Gen(pozice, slovo, zásobník, stav){ // pozice je vzdálenost od kotvy
  if stojím na neprázdném poli
    L = znak umístěný na poli, kde stojím
    GoOn(pozice, L, slovo, zásobník, NextArc(arc,L),stav)
  else if zásobník není prázdný
    forall znak L ze zásobníku, povolené na tomto poli
      GoOn(pozice, L, slovo, zásobník – L, NextArc(stav,L), stav)
```

```
    if zásobník obsahuje žolíka
      forall znak L, povolený na tomto poli
        GoOn(pozice, L, slovo, zásobník – žolík, NextArc(stav,L), stav)
}

GoOn(pozice, L, slovo, zásobník, nový_stav, starý_stav){
  if pozice < 0 // jsme v prefixu
    slovo = L + slovo
    if L ukončuje slovo ve starém_stavu a nalevo není položen znak
      Ulož_tah
    if nový_stav ≠ 0
      if nejsem na levém okraji desky
        Gen(1, slovo, zásobník, nový_stav)
      nový_stav = NextArc(nový_stav, @)
      if nový_stav ≠ 0, nalevo není znak a nejsem na pravém okraji desky
        Gen(1, slovo, zásobník, nový_stav)
  else if pozice > 0 // jsme v sufixu
    slovo = slovo + L
    if L ukončuje slovo ve starém_stavu a napravo není položen znak
      Ulož_tah
    if nový_stav ≠ 0 a nejsem na pravém okraji desky
      Gen(pozice + 1, slovo, zásobník, nový_uzel)
}
```

Algoritmus 4.1: Vyhledání všech tahů pro zadanou kotvu

Tahy jsme nalezli všechny a v následující kapitole se zaměříme na výběr tahu.

Kapitola 5

Herní strategie

V tomto oddíle rozebereme různé přístupy k výběru tahu. Herní strategie může být jednoduchá, například hladová strategie, která zvažuje pouze hodnotu tahu, ale může být i složena ze souboru mnoha rozdílných strategií, které se více a více specializují na řešení konkrétních podproblémů, nebo naopak vznikají i strategie pro výběr vhodné strategie. Příkladem takto rozsáhlého systému výběru tahu je systém Maven [6], vyvíjený již déle než 20 let.

Herní strategie mohou být děleny podle různých kritérií. Například dle herních aspektů, které zvažují. Zde se objevují strategie beroucí v potaz dostupnost bonusových polí na desce, kvalitu a vzájemnou provázanost zbylých písmenek v zásobníku, využití žolíka, neumožnění soupeři zahrát tah s 50ti bodovým bonusem a mnohé další. Herní strategie dále můžeme dělit dle způsobu, jakým získávají své vstupy. Například strategie pracující nad slovníkem, takové strategie provádějí různé statistické rozbory slov obsažených ve slovníku, například četnosti jednotlivých písmen a dvojic nebo i trojic písmen, často se vyskytující prefixy a sufixy atd. V poslední době se také stávají oblíbené strategie pracující nad stavovým prostorem, například Monte Carlo sampling [6], které získávají svoje data ze statistiky náhodných průchodů stromem hry. Časté jsou pak strategie nad deskou, sledující například míru uzavřenosti hry.

5.1 Hladová strategie

Hladová strategie je z používaných strategií nejjednodušší, ale přesto vzhledem k bohaté slovní zásobě počítače pro běžného hráče představuje výzvu. Hladová strategie také slouží jako zajímavý etalon pro testování strategií pokročilejších.

Hladová strategie nemá prakticky žádné další nároky na výpočetní výkon, stačí pouze jedenkrát přečíst všechny tahy a vybrat z nich ten s nejvyšší bodovou hodnotou.

5.2 Vážená strategie

Vážená strategie představuje metodu, jak získat nějaký výsledek ze vstupů mnoha roztodivných heuristických funkcí pracujících nad různými aspekty hry. Prakticky se jedná o nastavení určité váhy pro každou z takových funkcí a jednotlivé tahy pak mohou být setříděny dle hodnoty odpovídající součtu řady, jejíž členy jsou produkty váhy a výstupu odpovídající heuristické funkce. Netriviální je otázka, jak takové váhy nastavit.

Úvodní hodnoty bývají zadány člověkem, jejich optimalizace už je výrazně náročnější. Optimalizace může být prováděna například učením se na již odehraný partiích špičkových hráčů [7].

5.3 Monte Carlo sampling

Na metodu Monte Carlo lze narazit v mnoha vědních oborech. Nalézá uplatnění při modelování systémů obsahujících vysoké množství nejistoty, jakými jsou například fyzikální modely chování kapalin a plynů. V numerické matematice je využívána při integraci funkcí více proměnných.

Uplatnění této metody ve Scrabble zaznamenalo první úspěch v roce 1998, kdy program Maven [6], porazil světového šampióna ve Scrabble. Algoritmus 5.1 ilustruje použití metody Monte Carlo sampling v programu Maven.

```
Vytvoř množinu kandidátních tahů M
Pro všechny tahy T z množiny M
  Simuluj odehrání tahu T
  Proveď 20000 iterací
    Vyber náhodný zásobník protihráče
    Ulož nejhodnotnější tah protihráče pro vybraný zásobník
Zvol tah s nejméně hodnotnými tahy protihráče
```

Algoritmus 5.1: Monte Carlo sampling v programu Maven

5.4 Genetické algoritmy

Myšlenka využití přírodního evolučního procesu coby výpočetního nástroje se poprvé objevila v roce 1960 v práci Ingo Rechenberga [8]. V současnosti jsou genetické algoritmy používány v mnoha odvětvích lidské činnosti. Můžeme na ně narazit například při optimalizaci nakládání kontejnerů nebo i v predikcích akciových trhů.

Základní pojmy, se kterými se při použití genetických algoritmů setkáme, jsou:

- Jedinec – představuje řešení problému
- Hodnotící funkce – obvykle heuristická funkce podávající informaci o kvalitě jedince
- Křížení – obecný název pro postup, jak z jedinců jedné generace získat jedince generace následující

- Mutace – náhodný proces s předem danou pravděpodobností pozměňující vlastnosti jedince

- Generace – množina jedinců jedné iterace genetické algoritmu

Algoritmus 5.2 představuje ve zjednodušené podobě příklad průběhu genetického algoritmu.

```
Vytvoř počáteční populaci P
Ohodnoť kvalitu všech jedinců z P
Opakuj dokud není splněna podmínka ukončení
    Vyber novou generaci P' z P
    Proveď křížení nad P'
    Do P dosaď P'
    Ohodnoť kvalitu všech jedinců z P
Vrať nejkvalitnějšího jedince
```

Algoritmus 5.2: Představa jednoduchého genetického algoritmu

Podmínkou ukončení algoritmu může být například dosažení určité minimální kvality jedincem nebo provedení jistého počtu iterací.

Pro výběr nové generace je používáno mnoho technik odvíjejících se od konkrétního případu použití genetického algoritmu. Obvyklou metodou je výběr dle pořadí daného kvalitou jedinců s inverzně proporcionální pravděpodobností výběru, neboli čím vyšší je kvalita jedince, tím je pravděpodobnější, že bude vybrán.

Křížení jedinců může být realizováno náhodným výběrem vlastností dvou jedinců a vytvořením jedince nového z těchto vybraných vlastností. V závislosti na problému ale není vyjímečné provádět výběr vlastností z většího množství jedinců. Při vzniku nových jedinců pak přichází na řadu mutace, která s jistou pravděpodobností může zanést změnu některých vlastností.

Genetické algoritmy lze využít pro problémy jinak těžko řešitelné, jakými jsou například některé NP-úplné problémy. Vždy poskytnou nějaké řešení a umožňují vysokou míru paralelizace. Na druhou stranu většinou nemáme záruku, že výsledné řešení je optimální a nastavení některých parametrů, jako jsou pravděpodobnosti a míry mutace, metody výběru a mechanizmy křížení, mohou být značně netriviální.

Kapitola 6

Implementace

V této kapitole budou postupně probrány vybrané přístupy k implementaci systému realizujícího hru Scrabble. Podrobnosti jsou k nalezení ve vývojové dokumentaci.

Program byl implementován ve dvou fázích. Nejdříve bylo implementováno herní jádro, představující algoritmy pro vyhledávání všech tahů, heuristiky a genetický algoritmus pro jejich vyladění. Ve druhé fázi bylo implementováno grafické uživatelské prostředí a jeho napojení na herní jádro. Herní jádro programu je napsáno v C++ s využitím standardních knihoven, grafické uživatelské prostředí pak ve Visual C++ s využitím .NET.

6.1 Slovník

Slovník pro český Scrabble jsem převzal z internetové stránky [9] zabývající se herními strategiemi ve Scrabble. Kompletní oficiální slovník českého Scrabble v současnosti neexistuje, existuje pouze oficiální seznam dvou až pěti písmenných slov a oficiální ale komerční slovník Novex [1], obsahující slova do délky devíti písmen. K programu jsou přiloženy dvě varianty českého slovníku. Úplná verze se slovy délky až 15 písmen a odlehčená verze se slovy do délky 8 písmen.

Implementace slovníku je nezávislá na konkrétním jazyce. Ještě před konstrukcí slovníku dojde k promítnutí všech unikátních znaků obsažených ve slovníku do vnitřní reprezentace znaků a s touto reprezentací pak pracují veškeré další algoritmy. Zjednoduší se tak práce s pamětí a indexováním (znaky jsou reprezentovány přirozenými čísly 0..n-1, kde n je počet unikátních znaků) a program se stává na jazyce nezávislý. Z vnitřní reprezentace jsou znaky převáděny zpět až při vykreslování v uživatelském prostředí. Platí zde ovšem jistá omezení. Při použití jazyka s abecedou delší jak 41 znaků je třeba program znovu zkompilovat (podrobnosti jsou k nalezení ve vývojové dokumentaci), takových jazyků ale naštěstí mnoho není. Maximální počet znaků, který již nelze dále zvyšovat, je 255. Další omezení pak představuje vykreslování v uživatelském prostředí, kde musí být použité znaky vykreslitelné ve znakové sadě Microsoft Sans Serif.

Přiložen je i slovník pro anglický Scrabble, podrobnosti k realizaci změny slovníku jsou k nalezení v uživatelské dokumentaci.

K uložení slovníku byla zvolena datová struktura nazývaná GADDAG[5], která byla představena ve čtvrté kapitole. Volba této struktury byla dána její vysokou efektivitou při použití pro vyhledávání tahů. Vyšší paměťové nároky byly považovány za rozumnou cenu urychlení vyhledávání tahů, které má opodstatnění pro další použité techniky, za-

tímco ušetření paměti při současném růstu paměťových kapacit příliš nepřinese. Pro počítače s nižší dostupnou operační pamětí byl pro jistotu přidán redukovaný slovník, který by ani neměl představovat přílišné omezení, neboť tahy delší jak 8 písmen se ve hře vyskytují zřídka. Dalším důvodem volby GADDAG bylo, že naprostá většina současných implementací Scrabble používá DAWG [4] (představen ve čtvrté kapitole) a navíc pro češtinu nebyla autorem práce nalezena žádná implementace využívající GADDAG.

6.2 Algoritmus pro vyhledání všech tahů

Pro vyhledání všech tahů byl implementován algoritmus popsaný ve čtvrté kapitole. Transpozice hrací desky byla řešena zobecněním pohybu doleva a nahoru na pohyb zpět a pohybu doprava a dolu na pohyb dopředu. V části algoritmu, kde probíhá vyhledání všech tahů pro zadanou kotvu, byl přidán do rekurzivní funkce další parametr, který indikuje, zda algoritmus pracuje horizontálně nebo vertikálně, a tento příznak je návazně používán při dotazech na jednotlivá pole hrací desky. Implementovaný algoritmus je podrobněji popsán ve vývojové dokumentaci.

6.3 Heuristické funkce a herní strategie

Při výběru vhodných heuristických funkcí se vycházelo z předpokladu, že by bylo příjemné, kdyby stejně jako je implementace slovníku nezávislá na jazyce byly i heuristické funkce na jazyce nezávislé. Zaměřme se tedy na strategické invarianty, které jsou společné všem lokalizacím hry Scrabble.

Již od vzniku původního anglického Scrabble platí, že hodnota písmen je nepřímo úměrná jejich uplatnitelnosti ve hře. Tedy čím dražší písmeno, tím hůře se nám bude pro něj hledat použitelné slovo. V důsledku tedy čím dražší písmena budeme mít v zásobníku, tím méně se nám jich podaří uplatnit. Na druhou stranu, dražší písmena, ačkoli jich uplatníme méně, nám mají šanci přinést větší zisk. Můžeme tedy říci, že dražších písmen je vhodné se zbavovat, abychom měli větší výběr tahů, ale zase není dobré se jich zbavovat příliš levně. První heuristická funkce je postavena právě nad využíváním drahých písmen. Její výstupní hodnota pro daný tah je rovna součtu hodnot drahých písmen, které jsou součástí tahu, mínus součet hodnot drahých písmen, které zůstávají v zásobníku. Za drahé písmeno považujme písmeno hodnoty tři a více. Popsanou funkci označíme h_1.

Žolík je velmi podstatným herním kamenem a jeho výhodné uplatnění může častokrát výrazně napomoci k vítězství. Otázkou tedy zůstává, jak poznat, kdy už nadešla chvíle pro jeho uplatnění. Kdybychom čekali na výhodný tah dlouho, pak budeme mít po tuto dobu blokovanou pozici v zásobníku. Nad žolíkem je postavena heuristická funkce, kterou označíme h_2 a jejímž výstupem bude polovina počtu žolíků zůstávajících v zásobníku mínus počet žolíků uplatněných v tahu.

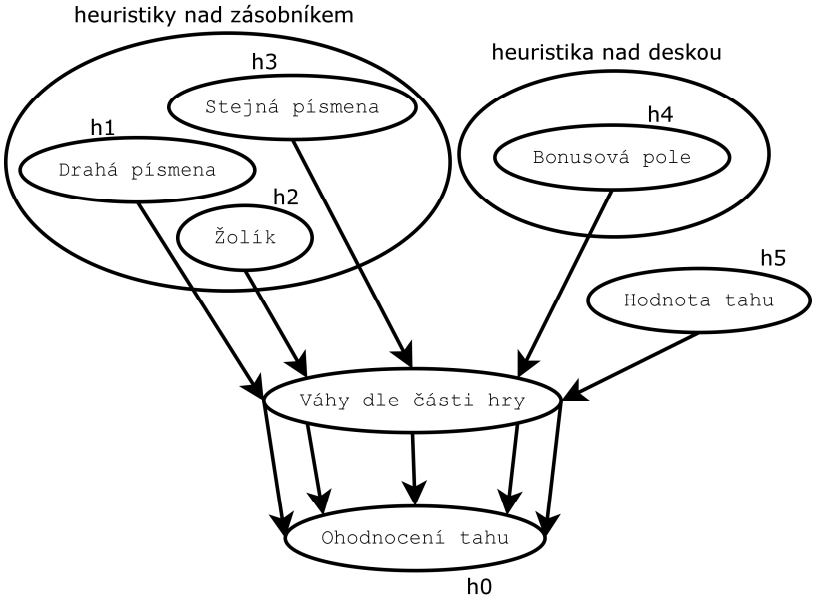
Výběr hratelných tahů může být značně snížen opakovaným výskytem stejných písmen v zásobníku. Postavíme tedy nad zásobníkem heuristickou funkci, jejímž výstupem bude počet písmen, které se v zásobníku vyskytují vícekrát, a označme ji h_3.

V průběhy hry dochází prakticky neustále k boji o bonusová herní pole, jejichž bonus se započítá pouze hráči, který na ně položil písmeno. Opakované zpřístupňování bonusových polí protihráči zřejmě nebude výhodné. Heuristickou funkci, jejíž výstupem bude ohodnocení všech zpřístupněných bonusů daným tah, beroucí v potaz velikost bonusu a jeho dostupnost ze zahraného tahu, označíme h_4.

Poslední funkci, která v sobě žádnou heuristiku neukrývá, označíme h_5 a jejím výstupem bude bodová hodnota tahu.

Navržené heuristiky jsou závislé na stavu, ve kterém se hra nachází. Proto ještě rozdělíme hru na pět částí. První část hry bude představovat zahájení, tedy položení prvního tahu hry. Pokud v sáčku již nezbývají žádná písmena, pak řekneme, že jsme v poslední části hry. Zbývající tři části hry budou dle počtu odehraných kamenů rovnoměrně pokrývat hru mezi jejím zahájením a vyprázdněním sáčku.

Pro získání konečného ohodnocení tahů, dle kterého už můžeme vybrat nejvýhodnější tah, použijeme váženou strategii, jejíž váhy se navíc budou měnit v závislosti na části hry, ve které se nacházíme. Funkci realizující zvážení výstupů heuristických funkcí označíme h_0. Obrázek 6.1 schematicky znázorňuje průběh ohodnocení tahu.



Obrázek 6.1: Heuristické funkce

K optimalizaci jednotlivých vah byl použit genetický algoritmus.

6.4 Genetický algoritmus

Genetický algoritmus (dále jen GA) coby optimalizační metoda autora práce zaujal natolik, že se ho rozhodl využít pro optimalizaci vah v použité vážené strategii.

Jedinci v GA jsou představováni počítačovými hráči Scrabble. Všichni hráči sdílejí společnou váženou strategii, ale každý hráč má svoje váhy, které jsou ve strategii použité pro výběru tahu. Tyto váhy představuji vlastnosti hráče, jejichž optimalizací se GA zabývá. Pro každou část hry je dána váha pro každou heuristickou funkci, celkově je tedy hráč popsán souborem 25 vah. Jelikož nám ale jde o poměr těchto vah, pak můžeme zvolit jednu váhu pevně a ta se již v průběhu algoritmu měnit nebude. Za pevnou váhu byla zvolena váha odpovídající funkci bodově hodnotící provedený tah. K optimalizaci pomocí GA tedy připadá 20 hodnot vah.

Navržení funkčního GA představovalo opakované zkoušení různých způsobů křížení, pravděpodobnosti křížení a pravděpodobni mutace. Častokrát docházelo, při zvolení nedostatečně velké populace nebo nevhodném nastavení mutace, k degeneraci hráčů, kteří již po pár generacích nebyli schopni ani „položit kámen“. Hlavním problémem bylo, jak hodnotit kvalitu hráčů a jak obejít výskyty statistických anomálií, deformujících hodnocení kvality.

Ze začátku bylo ponecháno v každé generaci odehrát každého hráče s každým jednu hru a kvalitu hráče představoval počet vítězství. Tento postup se ovšem neosvědčil, neboť hráči ztráceli během generací na kvalitě a referenční utkání s hladovou strategií byla horší a horší. Druhým důvodem byla vyšší výpočetní náročnost, neboť pro každou generaci bylo třeba odehrát $(n^2 – n) / 2$ partií. Přidání hráče používajícího hladovou strategii, který vznikl vynulováním všech vah kromě váhy odpovídající bodové hodnotě tahu, do každé generace znehodnotilo rozvoj hráčů, neboť hladová strategie byla častokrát dominantní a v důsledku tedy často křížena.

Verze GA, která se osvědčila, nechává každého hráče jedné generace odehrát jistý počet her proti pevně danému hráči používajícího hladovou strategii. Použitá verze algoritmu vypadá následovně:

```
Vytvoř novou náhodnou generaci hráčů G velikosti 100
Opakuj do nekonečna
  Pro každého hráče z G odehraj 30 her proti hladové strategii
  Setřiď hráče sestupně dle počtu zaznamenaných vítězství
  Vytvoř prázdnou generaci G'
  Dokud není naplněna generace G'
    Vytvoř nového hráče H
    Pro každou část hry
      S pravděpodobností 20%
          Vyber hráče z G s pravděpodobností úměrné počtu výher
          Dosaď do H váhy vybraného hráče
          S pravděpodobností 8% proveď mutaci dosazených vah
      S pravděpodobností 80%
          Vyber hráče z 10% nejkvalitnějších v G
          Dosaď do H váhy vybraného hráče
          S pravděpodobností 8% proveď mutaci dosazených vah
```

```
    Zařaď hráče H do generace G′
  Dosaď G′ do G
```

Algoritmus 6.1: Podoba použitého genetického algoritmu

GA byl ponechán v běhu do nekonečna a průběžně byly sledovány zaznamenané výsledky jednotlivých generací. Častokrát bylo i zasahováno do jednotlivých generací. GA probíhal na počítači Intel P4 Northwood 3GHz, 1GB 400 Mhz RAM, MS Windows XP. Za jednu sekundu bylo v průměru odehráno 3,17 partií. Celkově bylo během ladění GA odehráno kolem dvou milionů her.

Hodnoty vah získané pomocí GA jsou uloženy v externím souboru, který je vždy načítán při spuštění programu. Tyto hodnoty pak při hře využívají všichni hráči ovládaní počítačem.

6.5 Grafické uživatelské rozhraní

Propojení herního jádra s grafickým uživatelským prostředím (dále jen GUI) je realizováno pomocí ukazatele z GUI na třídu herního jádra. Veškeré stěžejní herní algoritmy jsou volány přes tento odkaz a v GUI pracují pouze algoritmy zajišťující korektní reprezentaci výsledků z herního jádra.

Kapitola 7

Zadání a zhodnocení

V této kapitole bude připomenuto zadání praktické části práce a autor práce se pokusí shrnout, do jaké míry se mu podařilo dosáhnout zadaných cílů.

7.1 Zadání

Cílem projektu bude vytvořit program umožňující hru Scrabble, a to buďto hru několika hráčů proti sobě nebo jednoho hráče proti počítači.

Projekt se zaměří zejména na strategii hry počítače. Musí umět v přiměřeném čase na rozumném počítači najít bodově nejhodnotnější tah a v koncovce, kdy jde o hru s úplnou informací, najít optimální strategii.

V části hry bez úplné informace je cílem navrhnout a implementovat strategii, která bude lepší než pouhý výběr bodově nejhodnotnějšího tahu.

Student si zvolí vhodný programovací jazyk pro implementaci tohoto problému.

7.2 Zhodnocení

Vytvořený program umožňuje hrát Scrabble libovolné kombinaci skutečných a počítačem ovládaných hráčů. Platí pro Scrabble obvyklé omezení na maximální počet čtyř hrajících hráčů.

Implementované řešení nachází na současných počítačích všechny proveditelné tahy hráče ve zlomcích vteřiny a uplatněná strategie tento čas výrazně nenavyšuje. V případě držení jednoho nebo dvou žolíků v zásobníku již může uživatel zaznamenat jistou prodlevu, ta ale nepřesahuje řádově jednotky sekund. Nelze ovšem zaručit, že hra bude stále pružně reagovat, pokud uživatel v uložené hře zvýší počet držených žolíků.

Autorovi práce se nepodařilo navrhnout, a ani v dostupných pramenech nalézt metodiku, která by umožnila najít v koncové části hry optimální strategii. Ačkoli se po vyprázdnění sáčku jedná o hru s úplnou informací, stavový prostor, který by bylo nutné prohledat, je stále příliš rozsáhlý pro využití tradičních metod, například minimaxu s α-β prořezáváním, který nalezení optimální strategie umožňuje. Během hry dochází v průměru k nalezení 238 tahů a v závěrečné fázi je průměr počtu nalezených tahů 76 (statistiky získané simulací jednoho tisíce náhodných her).

Implementovaná strategie poráží hladovou strategie statisticky ve více jak 63 % her (testováno na vzorku jednoho tisíce náhodných her) a měla by tak představovat výzvu

i pro zkušeného hráče Scrabble. Průběh každé partie je zaznamenáván. Uživateli je umožněno kdykoli během hry uložit záznam partie, který obsahuje všechny odehrané tahy a všechny tahy, které by bylo možné v daných situacích odehrát s použitím hladové nebo vážené strategie. Tento záznam také umožňuje sledovat, kdy by bylo rozhodnutí o výběru tahu váženou strategií prospěšnější než volba nejhodnotnějšího tahu. Výstupním formátem záznamu hry je HTML, umožňující přehledné procházení herních situací pomocí webového prohlížeče. Uživateli je také umožněno dříve uloženou nedohranou partii načíst a pokračovat v ní.

Jazykem pro implementaci herního jádra byl zvolen jazyk C++ pro jeho rychlost a případnou přenositelnost. Implementace grafického uživatelského rozhraní proběhla s využitím .NET v jazyce Visual C++, který usnadňuje tvorbu grafických aplikací díky množství předdefinovaných komponent.

Kapitola 8

Závěr a směry dalšího vývoje

V této práci byla představena problematika tvorby systémů implementujících hru Scrabble a v kapitolách teoretické části byly nastíněny možné přístupy k realizaci jednotlivých částí těchto systémů. V praktické části byla popsána implementace vybraných přístupů.

Kromě samotného umožnění hrát Scrabble v elektronické podobě je přínosem vzniklého programu například možnost přenesení hry Scrabble i do jazyka, pro který desková hra Scrabble dosud lokalizována nebyla. Stačí mít pouze slovník daného jazyka v textové podobě a navrhnout hodnoty a počty unikátních písmen ze slovníku dle počtu jejich výskytů ve slovníku. Program samotný je již na jazyce nezávislý a uživatel tak může hrát proti počítačovému hráči, který není použitým jazykem výrazněji omezený. Program je možné dále využít pro testování nových strategických přístupů a heuristik.

Program je možné v budoucnu dále rozvíjet doplňováním nových heuristik a herních strategií a případným vylepšováním datové struktury slovníku pro snížení její paměťové náročnosti. Grafické rozhraní by si také zasloužilo nějaké zkrášlení a případné ozvučení by mohlo zvýšit atraktivitu programu pro uživatele. Pro využití programu v jiných jazycích by také bylo vhodné vytvořit oddělený jazykový modul, který by umožnil přepínat jazyk, prostřednictvím kterého dochází k interakci s uživatelem.

Pro odlišení od jiných implementací hry Scrabble byl program pojmenován Frabble.

Literatura

[1] Česká asociace Scrabble (ČAS) (2006): http://scrabble.hrejsi.cz.

[2] R. de la Briandais (1959): File Searching Using Variable Length Keys. *Proceedings of the Western Joint Computer Conference*, 295–298.

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

[4] Appel, A. W., Jacobson, G. J. (1988): The world's fastest Scrabble program, *Communication ACM* **31**, 572–578, 585.

[5] Gordon, S. A. (1994): A faster Scrabble move generation algorithm, *Software-Practice and Experience* **24**, 219–232.

[6] Sheppard B. (2002): World-championship-caliber Scrabble, *Artificial Intelligence* **134**, 241–275.

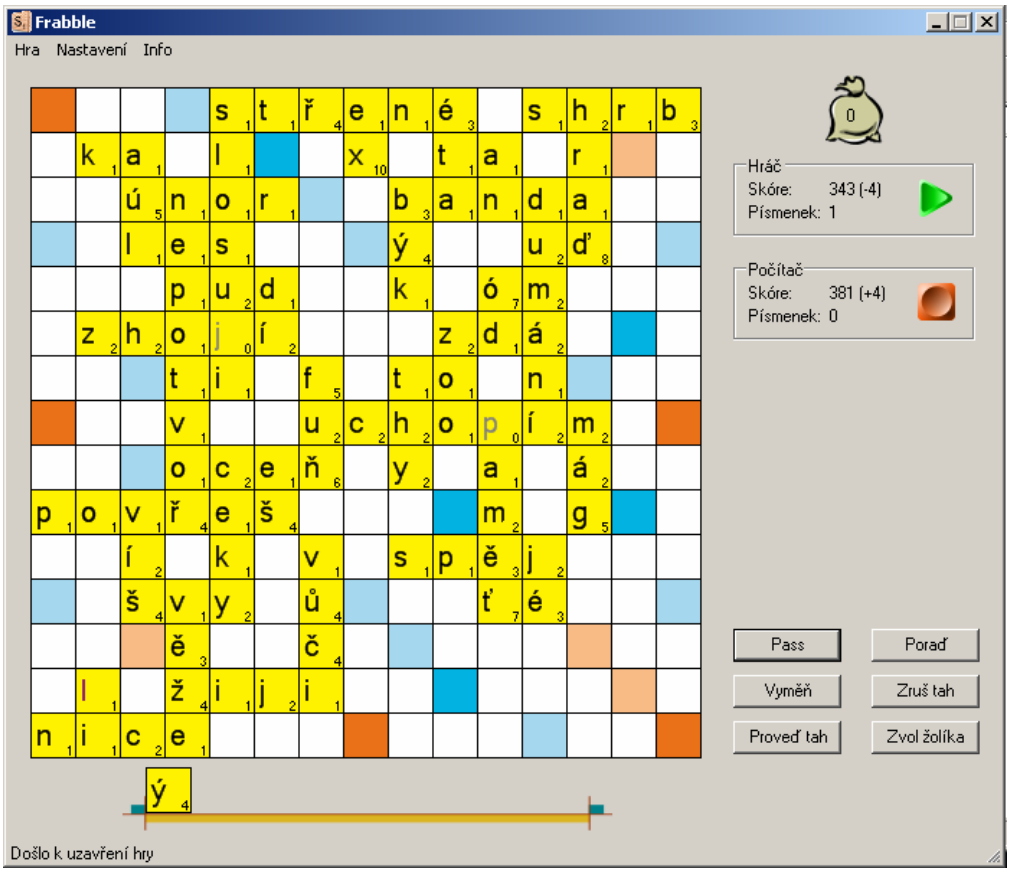
[7] Hotárek, V. (2003): Scrabble, diplomová práce, Fakulta informatiky Masarykovy univerzity Brno.

[8] Rechenberg I. (1960): Evolution strategies.

[9] Wordgame programmers – Scrabble (2006): http://www.gtoal.com/wordgames/scrabble.html.

Příloha A

Grafická podoba aplikace



Obrázek A.1: Ukázka grafické podoby aplikace