Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Jan Cipra

SCRABBLE

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr Jana Štanclová
Studijní program: Aplikovaná informatika

2006

Děkuji RNDr Janě Štanclové za pomoc a volnost při realizaci této práce.

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

V Praze dne 31.5.2006 Jan Cipra

Obsah

Název: SCRABBLE
Autor: Jan Cipra
Katedra: Katedra softwarového inženýrství
Vedoucí práce: RNDr Jana Štanclová
e-mail vedoucího: Jana.Stanclova@mff.cuni.cz
Abstrakt:

Cílem práce je nastudovat a implementovat algoritmy a struktury vhodné pro hru Scrabble. Studiem práce je seznámit se s možnostmi efektivního algoritmu určeného pro hraní hry počítačovým hráčem a s tím souvisejícího uložení slovní zásoby. Výsledný softwarový systém umožní hru více hraců po síti i hru, které se účastní počítačový hráči.

Klíčová slova: Scrabble, trie, konečné automaty

Title: SCRABBLE
Author: Jan Cipra
Department: Department of Software Engineering
Supervisor: RNDr Jana Štanclová
Supervisor's e-mail: Jana.Stanclova@mff.cuni.cz
Abstract:

The aim of this work is to examine and implement algotithms and data structures suitable for Scrabble game. The most important data structure is the lexikon of legal words. The program allows human players to play against each oder or computer controlled players. Moreover, the players can play over network.

Keywords: Scrabble, trie, finite state automata

1 Úvod

V poslední době se u nás i v celém světě projevuje stále větší zájem o stolní deskové hry. Kromě klasických zástupců deskových her, jako jsou šachy a dáma, se objevují nové varianty těchto her a jejich dostupnost se zvyšuje i u nás. Díky tomu, se pak do obecného podvědomí dostávají i zástupci méně tradičních her, jako například go. Jednou z dalších oblíbených her v anglicky mluvících zemích je již po několik desetiletí desková hra Scrabble. Její popularita je veliká a díky stálému rozšiřování i do jiných zemí dále roste. Oblíbenost této společenské hry spočívá v jednoduchosti pravidel a vhodné kombinaci jazykových schopností s prvky strategie. Mě k ní přivedl program s názvem Criss Cross, od českého vývojářského týmu Flash Software. Byla to právě tato hra, která mě inspirovala k vytvoření této bakalářské práce. Když jsem se s ní poprvé seznámil, okamžitě mě zaujala. Hrál jsem jí jak proti počítači, tak s přáteli. A právě tehdy mě mrzelo, že hra neobsahuje možnost hrát ji prostřednictvím síťového propojení. Rozhodl jsem se, že se o hře dozvím více. Prostuduji pravidla, možnost implementace počítačového hráče, možnost síťové komunikace a z toho všeho vytvořím tuto práci.

Cílem práce je tedy seznámení se s teorií vztahující se k implementaci počítačem řízeného hráče a následně hru naprogramovat tak, aby byla hratelná jak pro hráče začátečníky tak pro hráče středně zdatné. Druhým cílem je implementovat i možnost síťové hry.

V úvodní kapitole jsou pro ilustraci problematiky nastíněna základní pravidla hry, v další části uvedu možnosti ukládání slovní zásoby, poté zmíním algoritmus pro generování tahů a v poslední části představím výsledek mé praktické činnosti, tedy aplikaci Cross.

2 O Scrabble

2.1 Stručná historie

Kořeny této hry sahají do 30. let minulého století, konkrétně do roku 1931, kdy Alfreda Moshera Buttse napadla myšlenka vymyslet strategickou hru, která by měla za úkol procvičit jazykové schopnosti a zároveň by nebyla příliš složitá pro obyčejného člověka. 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. Všechny jeho tehdejší žádosti byly však 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 dodnes, stejně tak 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 na Scrabble. Konečně se také podařilo hru patentovat a od té doby je název zaregistrován jako ochranná známka společnosti Milton Bradley, divize Hasbro. Přes počáteční neúspěchy s prodejem hry se dostavily první kladné ohlasy. 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 v Londýně konalo první mistrovství světa ve Scrabble. 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. Více k historii je možné najít na [CAS].

Pokud se zaměřím na programy, které hru implementují, pak ze starších mohu jmenovat například program MONTY. Z těch novějších je to právě výše zmíněný program Criss Cross, který mě ke Scrabble přivedl.

2.2 Pravidla

Níže uvedená pravidla se týkají české verze hry Scrabble, tak jak byla schválena ČAS a jsou uvedena ve zkácené verzi. Nejprve vysvětlím několik pojmů:

- *Sáček s písmeny* – obsahuje hrací kameny, které představují jedno písmeno s příslušným ohodnocením
- *Zásobník* – obsahuje kameny, které si hráč vylosoval, měl by být skryt před zraky ostatních hráčů
- *Hrací plán* – plocha o standartní velikosti 15x15 hracích polí, na která hráči umísťují své kameny
- *Bonusová pole* – políčka hracího plánu, která znamenají pro hráče navýšení bodového zisku, pokud je na ně umístěn platný tah (viz část o bodování)
- *Startovací pole* – pole hracího plánu, přes které musí vést zahajovací tah

2.2.1 Začátek hry

Nejprve si všichni hráči vytáhnou ze sáčku s písmeny jeden hrací kámen. Začíná ten, který má kámen s písmenem nejblíže začátku abecedy. Poté se kameny opět vrátí do sáčku. Každý hráč si vylosuje sedm kamenů a uloží je do svého zásobníku tak, aby je ostatní neviděli. Začínající hráč vytvoří podle pravidel uvedených níže slovo ze dvou nebo více svých kamenů a položí ho na hrací desku tak, aby jedno z jeho písmen leželo na růžovém středovém poli. Kromě této podmínky se první tah řídí stejnými pravidly jako všechny následující (viz níže). Po skončení tahu jsou na řadě další hráči, počínaje hráčem po levici začínajícího.

2.2.2 Průběh hry

Hráč, který je na řadě, vybere ze svého zásobníku jeden nebo více kamenů a položí je na hrací desku. Potom si sečte body za všechna vytvořená či obměněná slova (viz kapitolu Bodování) a oznámí je zapisovateli. Považují-li jeho spoluhráči některé nově vzniklé slovo za *nepřípustné*, mohou ho zpochybnit před zahájením tahu dalšího hráče. Pokud je jedno z právě vytvořených slov uznáno podle níže uvedených pravidel za neplatné, vezme si hráč nazpět své kameny a ztrácí tah. Nakonec si dolosuje tolik kamenů, kolik jich použil při tvoření slov. Po skončení tahu tedy bude mít opět sedm kamenů.

Místo položení slova může hráč svůj tah využít k *výměně* některých či všech svých *kamenů*. Výměnu provede tak, že položí své kameny lícem dolů, vybere

z ostatních odpovídající počet nových kamenů a pak své staré kameny zamíchá mezi ty, ze kterých se losuje. V tomto kole již netvoří slova a na řadě je další hráč. Výměnu kamenů nelze provést, jestliže v sáčku zbývá méně než sedm kamenů. Pokud hráč nechce tvořit slova ani měnit kameny, může se *vzdát tahu*. Ostatním to oznámí tím, že řekne *pass*. Hráč se může tahu vzdát i v případě, že by byl schopen z písmen ve svém zásobníku utvořit slova.

2.2.3 Pokládání kamenů

Alespoň jeden z nově položených kamenů musí stranově sousedit s některým kamenem, který už na desce ležel. Úhlopříčné sousedství se nepočítá. Všechny nově položené kameny musí ležet ve stejném řádku nebo sloupci a nesmí mezi nimi být mezera, tj. musí existovat slovo, na kterém se všechny podílejí.

Nové slovo, tvořené nebo dotvořené položenými kameny, musí být čitelné zleva doprava nebo shora dolů a musí být povolené (viz níže). Na všech místech, kde se ho z boku dotýkají jiné kameny, musí také vzniknout povolená slova čitelná zleva doprava nebo shora dolů. Nová slova tedy mohou vzniknout následujícími způsoby:

- Přidáním jednoho či více kamenů ke slovu, které již na hracím plánu je.
- Umístěním slova kolmo k již existujícímu slovu na hracím plánu. Takto vzniklé slovo buď využívá jedno z písmen kolmého slova, nebo ho přidáním písmene na jeho začátek či konec modifikuje. Nové slovo může rovněž přemostit dvě či více již existujících slov.
- Položením celého nového slova rovnoběžně vedle již existujícího slova tak, že těsně sousedící kameny vytvoří rovněž smysluplná slova.

Prázdný kámen (*žolík*), lze použít místo kteréhokoli písmene včetně těch, která se na žádném kamenu nevyskytují (např. "W", "Q", "Ö"). Hráč však musí oznámit, místo kterého písmene byl použit. Toto písmeno zastupuje žolík až do konce hry, což může mít význam při přikládání kamenů dalšími hráči. Žádný kámen nelze přesunout, bylo-li s ním už jednou hráno.

2.2.4 Bodování

Za každé písmeno všech nově vytvořených nebo obměněných slov obdrží hráč takový počet bodů, který je na něm uveden. Počítají se tedy nejen kameny, které hráč v daném tahu přidal, ale i ty, které již na desce ležely. Písmena, která leží ve dvou nových slovech současně, se započítají opakovaně, a to včetně svých případných prémií (viz níže).

Některá pole hracího plánu jsou *prémiová*. Mají následující význam:

- Světle modrá pole zdvojnásobují hodnotu písmene, které je na nich položeno.
- Tmavě modrá pole ztrojnásobují hodnotu písmene, které je na nich položeno.
- Růžová pole zdvojnásobují hodnotu celého slova, které je na nich položeno.
- Červená pole ztrojnásobují hodnotu celého slova, které je na nich položeno.

Prémiové pole si započítá pouze hráč, který na něj kámen položil. Pokud hráč využije v novém slově kámen, který už dříve na prémiovém poli ležel, započítává si pouze základní hodnotu kamene. Prázdný kámen ležící na červeném poli, zdvoj- či ztrojnásobuje slovní skóre přesto, že kámen sám nemá žádné bodové ohodnocení. Jestliže v jednom slově hráč využije současně písmenné a slovní prémie, započítá se slovní prémie až nakonec. Využije-li hráč dvě slovní prémie v jediném slově, započítají se postupně obě. Lze tedy získat i čtyř- nebo devítinásobek bodové hodnoty slova.

Hráč, který v jednom tahu umístí všech sedm kamenů ze svého zásobníku, získá zvláštní prémii 50 bodů. Tato prémie se připočte k bodům získaným v dotyčném tahu až po započtení písmenných i slovních prémií.

Na konci hry je skóre každého z hráčů zmenšeno o hodnotu kamenů, které nepoužil. Pokud některému z hráčů nezbyl v zásobníku žádný kámen, k jeho skóre se přičtou hodnoty všech kamenů, které zbyly ostatním hráčům.

2.2.5 Konec hry

Hra končí, jestliže některý hráč využil všechny své kameny a nemůže si již vylosovat žádné další. Nikdo další už nesmí táhnout a hráčům se upraví skóre podle kamenů, které jim zůstaly v ruce - viz bodování. Hra ovšem nekončí, pokud již nelze losovat nové kameny, ale všem hráčům ještě nějaké zbývají v ruce a hráči s nimi umí táhnout. Teprve když se všichni hráči ve dvou po sobě jdoucích kolech vzdají tahu, hra skončí.

2.2.6 Implementace pravidel v aplikaci Cross

Mojí snahou bylo implementovat do programu výše zmíněná oficiální pravidla co nejvěrněji, ale i přes to jsem se v některých jednotlivostech poněkud odlišil. Tyto odlišnosti zde nyní uvedu.

První se týtá začátku hry, konkrétně v programu neprobíhá žádné losování začínajícího hráče, ale pořadí je určeno již při založení hry (více v *Uživatelské příručce*). Dále v programu neplatí to, že hráč po neplatném tahu ztrácí možnost svůj tah opravit. Je to proto aby nedocházelo k zbytečnému poškozování hráčů z důvodu omezené slovní zásoby aplikace. Poslední odlišností je pak to, že *žolík* lze nahradit pouze písmenem, které se nachází na některém hracím kamenu.

3 Možnosti uložení slovní zásoby

Tato kapitola se zaměřuje na představení používaných struktur pro uložení slovní zásoby aplikace. Ty jsou využity hlavně pro generování tahů, které je popsáno v následující kapitole. Souhrně se těmto strukturám říká *lexikony*, aby se odlišil význam termínu „slovník“. Lexikony jsou podmnožinou slovníků a většinou je možné si je představit jako seznam slov, která nejsou nijak vzájemně provázána a četnost slov se pohybuje v řádu statisíců. 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.

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

- Validace slov vytvořených člověkem
- Zdroj pro hledání slovních tvarů

Základní požadavky na datové struktury pro uchování lexikonu jsou následující:

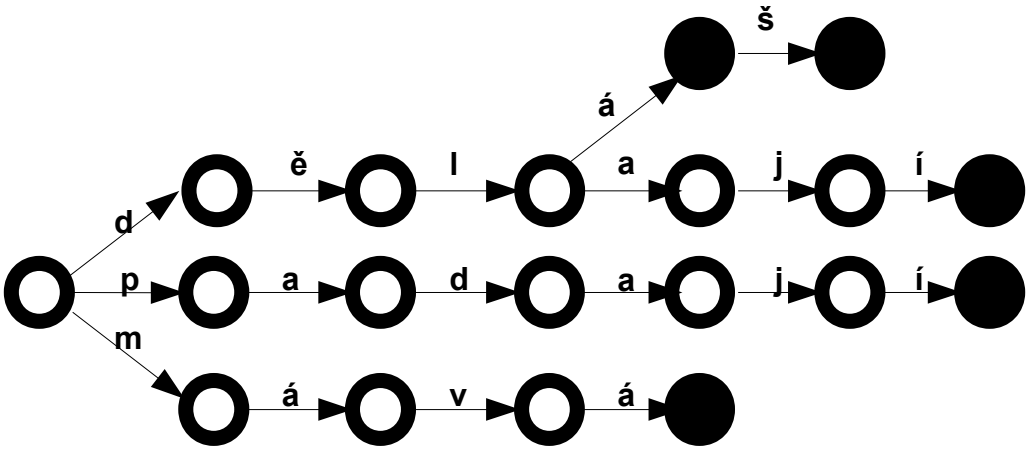
- *Paměťové nároky* – jak již bylo zmíněno, lexikony obsahují extrémní množství slovních tvarů, jejich velikost se může pohybovat v řádech desítek megabajtů. Práce s takto velkým objemem dat v paměti by znamenala nadměrnou paměťovou režii a vysoké nároky na výpočetní sílu. Proto je žádoucí objem dat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

Nyní se budu věnovat reprezentaci slovní zásoby, která byla použita pro imlementaci, a uvedu některé její vlastnosti s ohledem na vhodnost jejího použití v mé aplikaci.

3.1 Trie

Při bližším pohledu na slovní tvary z lexikonu je možné si všimnout mnoha podobností a pravidelností plynoucích z charakteru přirozeného jazyka. Například mnoho slov sdílí společnou předponu (díky skloňování) nebo příponu. Proto by bylo vhodné tyto společné části nějak ztotožnit.

Tento problém řeší stromové struktura nazvaná *Trie* [SED03]. Mějme abecedu Σ a množinu M slov nad touto abecedou. V trie vytvořené z množiny M pak každá cesta od kořene ke koncovému uzlu odpovídá právě jednomu slovu z M a výsledný strom je n-ární, kde n = |Σ|. Dále platí, že všechny listy jsou koncové. Pokud mají dvě slova společnou předponu, pak sdílejí společný úsek cesty ve stromu. Příklad trie pro množinu slov M = {dělá, děláš, dělají, padají, mává} je na obrázku 3.1, koncové uzly jsou černě vyplněny.



Obrázek 3.1: Trie pro množinu slov M = {dělá, děláš, dělají, padají, mává}

Vyhledávání slova S v trie pak probíhá následovně:

1. Ukazatel U se nastaví na kořen trie a symbol I na první znak S
2. Není-li v seznamu hran vycházejících z uzlu U hrana označená písmenem I, pak algoritmus vrátí NENALEZENO. V opačném případě se pokračuje dalším bodem
3. Po nalezené hraně se přejde do následujícího uzlu U'
4. Je-li I poslední písmeno hledaného slova a U' je koncový, pak se vrátí NALEZENO. V opačném případě se nastaví U na U' a I na další znak slova S a zopakuje se bod 2

Vyhledávání je velmi rychlé zejména při neúspěšném hledání, což je pro použitý algoritmus popsaný níže velmi výhodné. Díky velkému počtu stavů je paměťová složitost stále dosti velká a to i přesto, že dochází ke kompresi společných předpon. Nyní přijde na řadu konečný automat popsaný níže, který si poradí i se společnými příponami slov, a tím dále sníží paměťovou náročnost.

3.2 Konečné acyklické automaty

Deterministický konečný automat (DKA) M definujeme jako pětici $M = (Q,\sum,\delta,q_0,F)$ kde

- Q je neprázdná konečná množina stavů
- $\sum$ je neprázdná konečná množina symbolů, neboli abeceda
- δ: Q x $\sum \rightarrow$ Q je přechodová funkce
- $q_0 \in Q$ je počáteční stav
- $F \subseteq Q$ je množina koncových stavů

Navíc zavedeme rozšířenou přechodovou funkci δ^*: Q x $\sum^* \rightarrow$ Q, pro kterou platí

1. $\delta^*(q,\lambda) = q \quad \forall q \in Q \quad kde \ \lambda \ je \ prázdný \ symbol$
2. $\delta^*(q,wa) = \delta(\delta^*(q,w),a) \quad kde \quad q \in Q, w \in \sum^*, a \in \sum$

$\sum^*$ je množina všech řetězců nad abecedou $\sum$.

Jazyk rozpoznávaný automatem pak značíme L(M) a platí, že

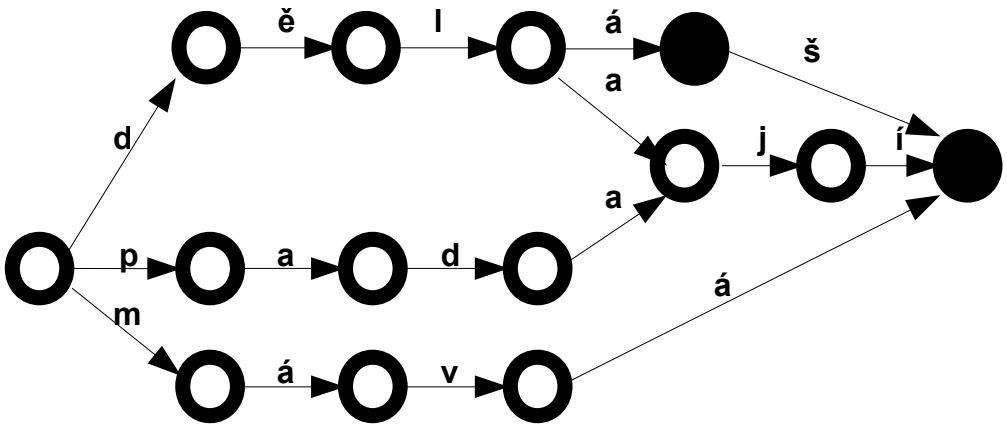
$$L(M) = \{ x \in \sum^*, \delta^*(q_0,x) \in F \}$$

Vztaženo ke Scrabble, jazyk L(M) odpovídá právě lexikonu. V další části bude představena metoda konstrukce minimalizovaného deterministického acyklického konečného automatu. Jeho využitím se sníží výsledná prostorová složitost, zatímco výhoda rychlého vyhledávání v těchto strukturách bude zachována. Nutnou a postačující podmínkou pro to, aby DKA byl acyklický, je konečnost L(M).

3.2.1 Minimalizace konečných automatů

Mějme deterministický konečný automat M. Minimální deterministický acyklický konečný automat ekvivalentní automatu M značíme jako *Min(M)* a platí, že počet stavů každého konečného automatu M' ekvivalentního s automatem *Min(M)* je větší nebo roven počtu stavů automatu *Min(M)*. Dva automaty jsou ekvivalentní pokud rozpoznávají stejný jazyk.

Na obrázku 3.2 je zobrazen minimální deterministický acyklický konečný automat ekvivalentní s trie z obrázku 3.1. Tradiční postupy minimalizace automatu probíhají tak, že se nejprve vytvoří deterministický konečný automat a poté je minimalizován. Já se zaměřím na minimalizaci, která probíhá současně s budováním automatu a využiji algoritmus publikovaný v článku [DAC98]. Díky inkrementální konstrukci, je algoritmus mnohem rychlejší a s menšími paměťovými nároky. To je částečně vykoupeno požadavkem na to, aby byla data před konstukcí lexikograficky uspořádána. Protože se slovník pro hru bude měnit pouze minimálně, nejedná se o přílišnou komplikaci.



Obrázek 3.2: Konečný automat ekvivalentní s trie z obrázku 3.1

Nyní následuje algoritmus na minimalizaci konečných automatů publikovaný v [DAC98].

```
Register := prázdná množina;
do další slovo na vstupu
      Slovo := další slovo;
      Předpona := Předpona(Slovo);
      DosaženýStav := δ*(q0,Předpona);
      Přípona := Slovo[length(Přípona) + 1...lenght(Slovo)];
      if Má_potomky(DosaženýStav)
            Odebrat_nebo_vyměnit(DosaženýStav);
      fi
      Přidat_příponu(DosaženýStav, Přípona);
od
Odebrat_nebo_vyměnit(q0);
```

```
func Předpona(Slovo)
      return Slovo[1...n]:n = max i: ∃q∈Q δ*(q0,Slovo) = q;
cnuf
```

```
func Odebrat_nebo_vyměnit(Stav)
      Potomek := Poslední_potomek(Stav);
      if not Registrováno(Potomek)
            if Má_potomky(Potomek)
                  Odebrat_nebo_registrovat(Potomek);
            fi
            if   ∃q∈Register(q≡Potomek)
                  Smazat_větev(Potomek);
                  Poslední_potomek(Stav) = q;
            else
                  Register :=   Register∪{Potomek}  ;
                  Registrovat(Potomek);
            fi
      fi
cnuf
```

Funkce *Předpona* najde nejdelší předponu slova, kerá je zároveň předponou slova přidaného dříve do automatu. Funkce *Přidat_příponu* přidá do automatu větev, která reprezentuje přidávané slovo (minimální příponu slova, která není předponou slova v automatu). Poslední stav této větve je označen jako koncový. Funkce *Poslední_potomek* vrátí ukazatel na lexikograficky poslední odkaz na další uzel vycházející z uzlu v argumentu. Každý uzel má příznak *Registered,* který značí, zda byl uzel již registrován či nikoliv. Hodnotu příznaku zjistíme pomocí funkce *Registrováno.* Funkce *Má_potomky* vrací true, když uzel obsahuje nějaké odkazy na potomky a funkce *Smazat_větev* smaže uzel v argumentu a všechny uzly z něj dosažitelné.

4 Generování tahů

Tato kapitola pojednává o nejdůležitější součásti implementace hry Scrabble, ve které může hráč změřit své síly s počítačovým protivníkem. Tedy o způsobu hledání všech možných platných tahů pro dané rozestavění kamenů na hracím plánu a hráčův zásobník.

Popsaný algoritmus je standardním algoritmem, který využívají aplikace hrající Scrabble, a to zejména díky své rychlosti. Jeho autory jsou A. Appel a G. Jacobson a návrh algoritmu pochází z roku 1988 [AJ88]. Hlavními rysy algoritmu jsou zjednodušení problému pomocí předpočítávaných množin a využití backtrackingu s vysokou efektivitou při tvorbě slovních tvarů pro jednotlivé tahy. Algoritmus neprohledává celý lexikon, ale prochází všechna pole na hrací ploše, kde je možno umístit kameny k písmenům dříve odehraným (mimo prvního tahu). Algoritmus 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 je použit konečný automat, díky své nízké paměťové náročnosti, snadnému a rychlému průchodu. Před popisem vlastního algoritmu budou definovány základní pojmy a struktury, jež jsou algoritmem používány.

4.1 Zjednodušení problému

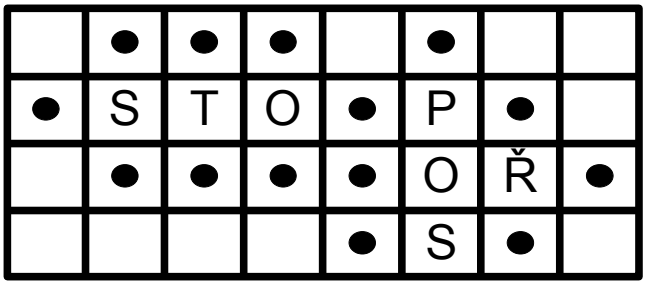
Slova se ve hře mohou tvořit *vodorovně* a *svisle* tak, že všechna nově položená písmena jsou v jednom řádku či sloupci. Generování tahů je možné zjednodušit tím, že se vytváří pouze vodorovné tahy. Tahy svislé se získají generováním tahů na transponovaném hracím plánu.

4.1.1 Množiny povolených písmen

Pokud je nové slovo pokládáno vodorovně, pak musí nově položené kameny, pokud jsou přímo nad nebo pod dříve položenými kameny, tvořit přípustné slovo i kolmo. Do každého sloupce lze takto přidat pouze jedno nové písmeno, a proto je možné pro všechna pole hracího plánu předpočítat množinu povolených písmen (těch, která po položení tvoří přípustné slovo). Tyto množiny je možné pro vodorovný a svislý směr spočítat před započetím generování tahů. Po položení nových kamenů stačí přepočítat pouze dotčenou část.

4.1.2 Kotvy

Každé nově položené slovo musí sousedit s již dříve položenými kameny. Proto hledání tahů začne na políčkách, která sousedí s již položenými kameny. Tato pole budeme nazývat *kotvou*. Na obrázku 4.1 vidíme část hracího plánu s vyznačenými kotvami.



Obrázek 4.1: Kotvy vyznačené na hracím plánu

4.2 Procházení

Při použití výše zmíněné pomocné množiny s povolenými písmeny je problém generování přípustných tahů pouze jednorozměrný. Pro každý řádek, hráčovy kameny, množiny povolených písmen a kotvy jsou vygenerována všechna přípustná slova pomocí jednoduchého algoritmu složeného z následujících dvou částí:

- Generování předpon
- Generování přípon

4.2.1 Generování předpon

Předponou se rozumí posloupnost písmen tvořená, písmeny z hráčova zásobníku nebo kameny již položenými na desce (nikoliv však jejich kombinací). Aby se zamezilo opakovanému vytváření stejných slov, nesmí předpona zasahovat do jiné kotvy.

Kotva je prázdné pole přímo sousedící s některým položeným kamenem. Protože předpona nesmí zasahovat do jiné kotvy a ani nesmí kombinovat nové kameny s již dříve položenými, je limitována její délka. Maximální délka přípony je rovna počtu prázdných políček neoznačených jako kotva bezprostředně zleva

napojených na příslušnou kotvu. Navíc protože kameny, které nejsou kotvami, mají triviální množinu povolených písmen, je při generování předpon důležitá pouze tato její maximální možná délka. Níže je uvedena funkce *Přepona* (parametr *Limit* je právě nalezená maximální možná délka), která volá pro každou vygenerovanou levou část funkci *Přípona*, jejíž popis je uveden v části 4.2.2.

```
func Předpona(ČástSlovo, Stav S, Limit)
      Přípona(ČástSlovo, Stav S, Kotva)
      if Limit > 0
            for each hrana H vycházející ze stavu S
                  if písmeno L reprezentující hranu H je mezi
                     kameny hráče
                        odebereme L z kamenů hráče;
                        N' je stav dosažený hranou H z N;
                        Předpona(ČástSlovo . L, N', Limit-1);
                        vrátíme L do kamenů hráče;
                  fi
            rof
      fi
cnuf
```

Symbolem „.“ je v algoritmu označeno připojení znaku na konec řetězce.

4.2.2 Generování přípon

Přípona (na rozdíl od předpony může být prázdná) musí obsahovat alespoň jeden znak a její utváření začíná na poli, které je označeno kotvou. Funkce *Přípona,* je volána pro každou vygenerovanou předponu a jako parametr dostává částečně utvořené slovo, stav ve slovníku, který byl dosažen při utváření předpony a kotvu.

Další prohledávání je následně omezeno na písmena ze zásobníku a ty, která se již nachází na desce. S jejich využitím mohou vznikat slova, která jsou spojením písmen nově a již dříve položených. Při průchodu se také využívá množin s povolenými písmeny pro jednotlivá políčka (viz 4.1.1), která zaručí že nově položené slovo utváří povolená slova i v kolmém směru.

Kdykoliv se v této fázi v automatu reprezentujícím slovník dosáhne koncového stavu, je příslušné slovo platným tahem a je pomocí funkce *PřidatTah* uloženo pro další zpracování.

```
func Přípona(ČástSlovo, stav S, pole P)
     if P je prádné
          if S je konečný stav
               PřidatTah(ČástSlovo);
          fi
          for each hrana H vycházející z S
               if písmeno L rep. hranu H je mezi kameny
                  hráče and L je mezi povolenými znaky
                    L se odebere z kamenů hráče;
                    N' je stav dosažený hranou H z N;
                    DalšíPole := pole vpravo od P;
                    Přípona(ČástSlovo . L, N', DalšíPole);
                    L se vrátí do kamenů hráče;
               fi
     else
          L := písmeno položené na poli P;
     if N má hranu H reprezentovanou písmenem L
               N' je stav dosažený hranou H z N;
               DalšíPole := pole v pravo od P;
               Přípona(Částlovo . L, N', DalšíPole);
          fi
     fi
cnuf
```

Nyní je algoritmus na generování tahů kompletní. Je tvořen dvěma rekurzivními funkcemi (*Přípona* a *Předpona*). Po provedení algoritmu jsou vydané tahy pomocí nějakého strategického postupu prohledány a jeden tah je vybrán jako „nejlepší“ (více v kapitole 5).

4.2.3 Poznámky

- *Prázdné prefixy* - V případě, že pole nalevo od kotvy slova (pro které je právě hledán tah) je obsazené odehraným kamenem, není utvářena žádná předpona. Místo toho se ihned volá funkce *Přípona* pro stav, který 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. Ve funkcích uvedených výše, není případ výskytu žolíku explicitně testován. Ošetření spočívá v přidání dodatečných testů 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 a opět nabude své původní podoby. 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 žolíků jedním hráčem je relativně malá.

5 Strategie výběru tahu

Problém stanovení tahu, který bude následně odehrán počítačem pro aktuální rozmístění hracích kamenů na ploše a dané hráčovy kameny, je v podstatě hlavním úkolem nejrůznějších heuristik. Standardně je nejdříve algoritmicky vygenerována množina všech přípustných tahů, které mohou bezprostředně následovat. Nad touto množinou jsou následně prováděny heuristiky, které 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 definice heuristické vyhodnocovací funkce, která každému nalezenému tahu přiřadí jeho „prospěšnost“ pro další hru, at’ již ve formě bodového ohodnocení či výhodných pozic na hracím plánu (bonusová pole). Důležitou vlastností hry Scrabble je, že jde o hru s takzvanou *neúplnou* informací což zásadním způsobem znesnadňuje použití strategií používaných například pro šachy. Neúplnost informace je způsobena tím, že hráči si tahají písmena ze sáčku náhodně a proto nelze žádným způsobem zjistit jeké kameny protihráč vlastní ani kameny, které budou vylosovány v pozdější fázi hry.

Proto jsem za strategii počítačového hráče zvolil Hladovou strategii (5.1), jež je jednoduchá a dle mého názoru pro účel mé aplikace plně dostačující.

5.1 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 jeví ve většině partií s průměrně zdatnými protihráči jako dostačující. V případě expertních hráčů není již tak efektivní, ale jak sem již zmínil pro účel této aplikace je dle mého názoru plně vyhovující.

Algoritmus uvedený níže nejprve ohodnotí všechny vygenerované tahy a poté vybere ten s nejvyšším ohodnocením.

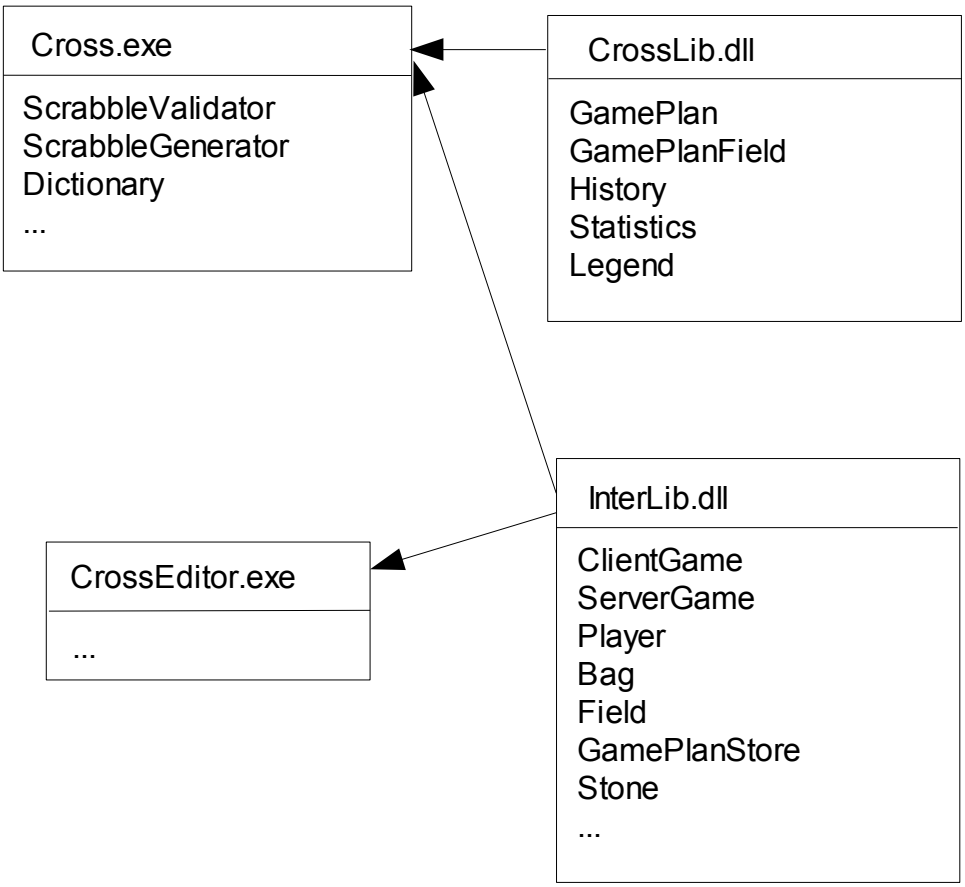
```
func VybratTah( Vygenerované tahy )
      for each vygenerovaný tah T
            OhodnotitTah(T);
      rof
      Vybere se tah největším ohodnocením;
cnuf
```

Funkce *OhodnotitTah* provede ohodnocení tahu. Volba ohodnocovací funkce byla v případě Scrabble velmi jednoduchá. Tahy se totiž ohodnotí bodovým ziskem, který je za jejich položení hráči připsán.

6 Implementace

Aplikaci Cross jsem vyvíjel v programovacím jazyce C++ .Net a to ve vývojovém prostředí Microsoft Visual Studio 2003. Při implementaci uživatelského a síťového rozhraní jsem použil knihoven .Net Framework.

Hra využívá dvou DLL knihoven. V té první s názvem *CrossLib.dll* jsou naprogramovány komponenty uživatelského rozhraní. Konkrétně to je panel s hracím plánem a tlačítky (*GamePlan* a *GamePlanField*), legenda s informací o barvě významných polí na hracím plánu (*Legend*), box obsahující informaci o hráčích a o zbývajícím čase (*Statistics*) a tabulka zobrazující historii tahů ve hře (*History*). Ve druhé knihovně s názvem *InterLib.dll* jsou třídy reprezentující například hráče (*Player*), hru (*ClientGame* a *ServerGame*) a třídy pro uložení dat do souboru. Obě knihovny jsou implementovány v jazyce C# a knihovnu *InterLib.dll* využívám také v Edidoru.

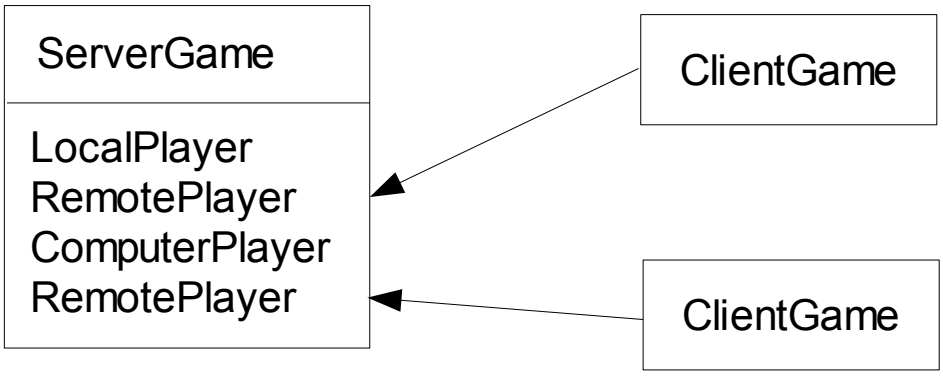


Obrázek 6.1: Hlavní knihovny a třídy aplikace

Asi nejdůležitější částí aplikace je Generátor tahů. Třída generátoru tahů se jmenuje *ScrabbleGenerator* a je v ní implementován algoritmus popisovaný v kapitole 4. Algoritmus využívá pomocnou strukturu pro uložení množin povolených znaků, pod názvem *CrossChecks.* Vygenerované nebo odehrané tahy jsou pomocí třídy *ScrabbleValidator* zkontrolovány. Tato třída také spočte počet bodů, které hráč za tah obdrží.

Další velmi důležitou součástí je slovník pro uložení slovní zásoby aplikace, která představuje asi 400 000 slov. Je implementován ve třídě *Dictionary.* Zde byl využit minimální deterministický acyklický konečný automat popsaný v části 3.2.

Komunikace po síti využívá TCP protokol a je implementována pomocí tříd z knihoven .Net Framework. Síťová hra je řešená tak, že hráč, který hru založí se chová jako server a ostatní jsou k němu připojeni v roli klientů. Po založení hry se čeká na připojení všech vzdálených hráčů. Poté se hráčům pošle zvolený hrací plán, sada kamenů, počet a jména hráčů a nastavení hry a hra se vytvoří. Pokud je na tahu hráč na serveru, nejprve se rozhodne o platnosti tahu a pak se platný tah rozešle všem vzdáleným hráčům. Pokud je na tahu vzdálený hráč, pak se jeho tah nejprve pošle na server. Tam se rozhodne o jeho platnosti a následně se v případě, že byl shledán platným rozešle všem vzdáleným hráčům. Pokud platný nebyl, pak je zpět hráči (tomu, který tah položil) posláno vyrozumění o jeho neplatnosti. Validace probíhá vždy na straně serveru proto, aby se pro kontrolu používalo shodného slovníku.



Obrázek 6.2: Schéma síťové hry

7 Uživatelská příručka

Ačkoliv je uživatelské rozhraní aplikace velmi jednoduché a intuitivní popíši v této kapitole jeho základní součásti a nakonec připojím ještě několik návodů jak co provést.

7.1 Instalace

Aplikace Cross ja spustitelná pod systémem Microsoft Windows XP s nainstalovanou podporou Microsoft .NET Framework (volně k dispozici na stránkách Microsoftu). Hra nemá žádné zvlášť vysoké nároky na výpočetní sílu ani paměťovou kapacitu počítače.

Na přiloženém CD je archiv, ve kterém po rozbalení do adresáře na disku je spustitelný soubor *Cross.exe*. Pomocí tohoto souboru se hra spouští.

7.2 Uživatelské rozhraní

Zde je uveden popis dialogů a hlavního okna aplikace.

7.2.1 Hlavní menu

Hlavní menu programu má dvě základní položky. První je položka *Hra*, která dále obsahuje volby:

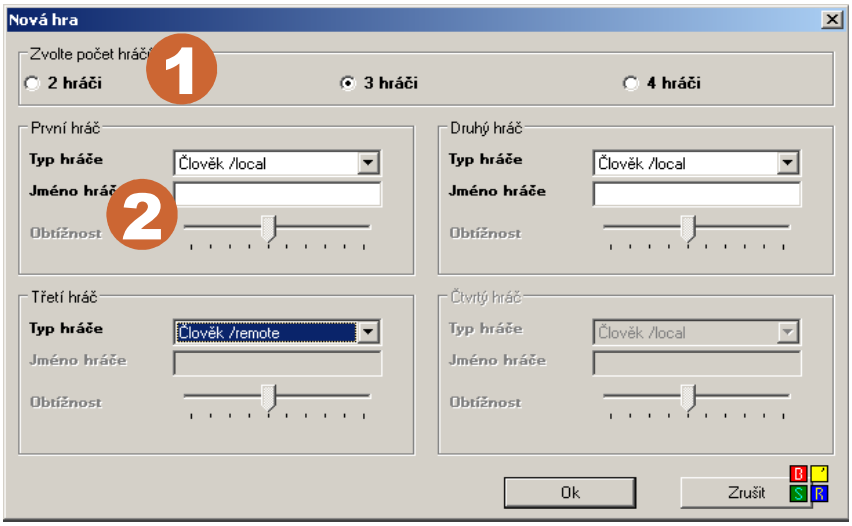
- *Nová hra* – zobrazí dialogové okno s jehož pomocí se založí nová hra (viz 7.2.2)
- *Připojit se ke hře* – umožňuje nastavení připojení na hru založenou na vzdáleném počítači (viz 7.2.3)
- *Ukončit hru* – ukončí právě rozehranou hru
- *Načíst hru* – zobrazí standardní dialog pro výběr souboru s již dříve uloženou hrou
- *Uložit hru* – zobrazí standardní dialog pro výběr umístění a názvu souboru na uložení právě rozehrané hry
- *Konec* – ukončí rozehranou hru

Druhou položkou je *Nastavení.* S její pomocí se zobrazí základní nastavení barevného vzhledu a způsobu hry (více v 7.2.4).

7.2.2 Dialog *Nová Hra*

Tento dialog (obr 7.1) umožňuje vytvoření nové hry. Dělí se na dvě části. V první části se volí pomocí přepínače počet hráčů. Ve druhé se pro každého z nich vyplní několik voleb:

- *Typ hráče* – zde máme na výběr ze tří možností
  - *Člověk /local* – označuje hráče, který bude hrát na počítači na němž byla hra založena
  - *Člověk /remote* – hráč, který se bude připojovat prostřednictvím sítě
  - *Počítač* – počítačem řízený hráč
- *Jméno hráče* – jméno, pod kterým chce příslušný hráč ve hře vystupovat
- *Obtížnost* – tento posuvník u počítačového hráče nastavuje jeho obtížnost. Směrem doleva se obtížnost snižuje a naopak

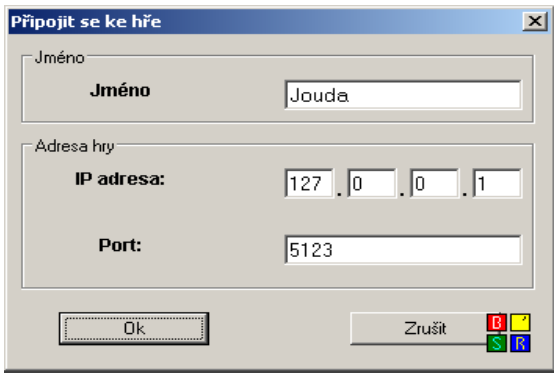


Obrázek 7.1: Dialog Nová hra

7.2.3 Dialog *Připojit se ke hře*

Pomocí dialogu *Připojit se ke hře* (obr. 7.2) je možné se připojit ke hře založené na vzdáleném počítači. Obsahuje tři pole:

- *Jméno* – jméno, pod kterým bude hráč ve hře vystupovat
- *IP adresa* – IP adresa stroje, kde běží hra, na kterou se chce hráč připojit
- *Port* – port, na kterém hra na cílovém počítači běží (standardně 5123)



Obrázek 7.2: Dialog Připojit se ke hře

7.2.4 Nastavení

V první záložce na tomto okně se pod názvem *Vzhled* (obr. 7.3) nastavují barvy polí na hracím plánu a kamenů.

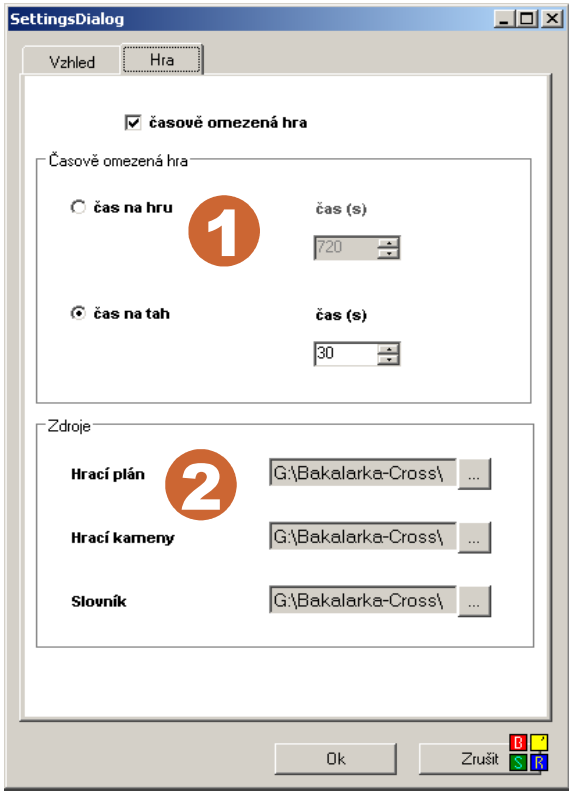
Barvy kamenů a polí hracího plánu se nastaví kliknutím na barevný obdélníček vedle příslušné popisky. Pomocí zobrazeného dialogu si hráč vybere požadovanou barvu.

Obrázek 7.3: Nastavení vzhledu

V druhé záložce pod názvem *Hra* (obr. 7.4) se nastavuje hra s omezeným časem (sekce 1) a také zdroj pro načtení hracího plánu, slovníku a hracích kamenů (sekce 2).

Pokud chce uživatel hrát hru s omezeným časem, nejprve zaškrtne políčko *časově omezená hra*. Poté si vybere, zda chce mít čas omezený na každý tah nebo mít limitovaný čas na celou hru. Nakonec v příslušném políčku nastaví časový limit v sekundách.

Pro modifikaci hry je možné změnit hrací plán, na kterém se hraje a také hrací kameny. K této změně slouží položky v části *Zdroje*. Zde se pomocí tlačítka a následně spuštěného dialogu zvolí umístění souboru s uloženým zdrojem. Ten může být například již dříve vytvořen v přiloženém editoru (viz část 7.3).



Obrázek 7.4: Nastavení hry

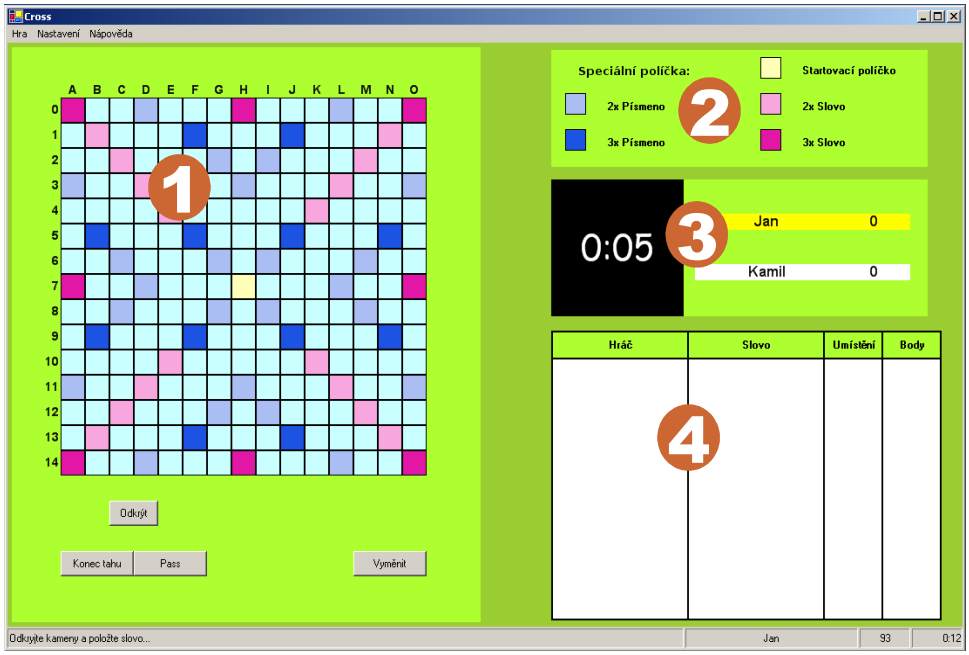
7.2.5 Hlavní okno aplikace

Hlavní okno aplikace (obr 7.5) má čtyři části:

- *Hlavní panel* (1) – na něm je zobrazen hrací plán, seznam hráčových kamenů a následující čtyři tlačítka:

- *Odkrýt* – na začátku tahu se pomocí tohoto tlačítka odkryjí kameny
- *Konec tahu* – ukončuje hráčův tah
- *Pass* – hráč se vzdává tahu
- *Vyměnit* – výměna kamenů místo položení slova

- *Legenda* (2) – legenda zobrazuje barvy a význam bonusových polí na hracím plánu
- *Statistika o hře* (3) – statistika zobrazuje jména a počty bodů všech hráčů a v případě že se hraje hra s časovým omezením, zobrazuje i údaj o zbývajícím čase
- *Historie tahů ve hře* (4) – zde se průběžně zobrazují odehrané tahy se jménem hráče, pozicí na hracím plánu („-“ znamená, že slovo bylo položeno vodorovně a „|“ svisle) a bodovým ohodnocením. Tah je zde zobrazen i v případě, že hráč měnil kameny, vzdal se tahu nebo mu vypršel čas na tah.



Obrázek 7.5: Hlavní okno aplikace

7.2.6 Stavový řádek aplikace

Stavový řadek obsahuje informaci o hráči, který je právě na tahu, počtu kamenů v sáčku, čas od počátku hry a také jednoduché kontextové informace.

7.3 Jak na to

Zde uvádím několik postupů jak pracovat s aplikací.

7.3.1 Jak založit novou hru

Pomocí volby v Hlavním menu *Hra→Nová hra...* vyvolám dialog (obr. 7.1), v něm zvolím počet hráčů a jejich vlastnosti a stisknu *Ok.* V případě, že jsem zvolil nějaké vzdálené hráče, počká se na jejich připojení, jinak se okamžitě zobrazí hrací plán a hra začne.

Pokud bych chtěl spustit časově omezenou hru, musím provést příslušné změny v nastavení (viz 7.2.4).

7.3.2 Jak se připojit ke hře

Pomocí volby v Hlavním menu *Hra→Připojit se ke hře...* vyvolám dialog(obr. 7.2), v něm zvolím jméno, IP adresu a port počítače, na který se chci připojit a stisknu *Ok.* Ve chvíli, kdy se ke hře připojí všichni vzdálení hráči, se hra zobrazí.

7.3.3 Jak hru Uložit/Načíst

Pomocí volby v Hlavním menu *Hra→Uložit* nebo *Hra→Načíst* a následně spuštěných dialogů. Ukládat lze pouze hru bez vzdálených hráčů.

7.4 Editor

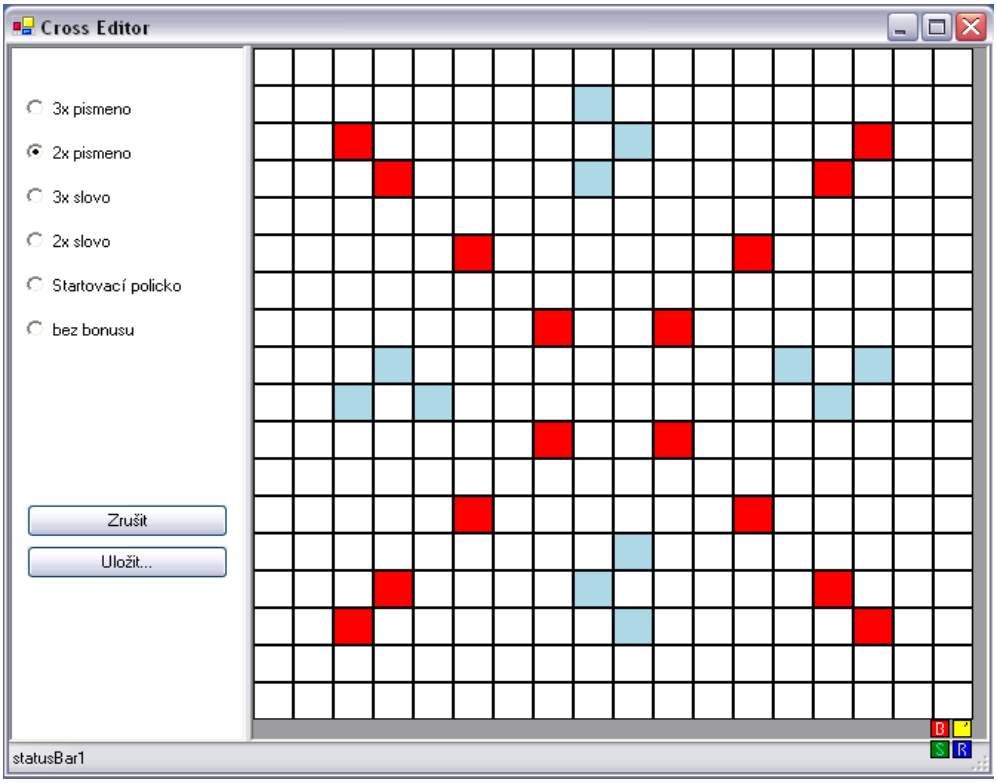
Ke hře je k dispozici také editor, pomocí kterého lze modifikovat nebo vytvářet vlastní hrací plány a seznamy hracích kamenů.

Editor se spouští souborem *CrossEditor.exe.* Po spuštění se zobrazí hlavní okno dialogu, na kterém jsou v levé části tři tlačítka. Pomocí tlačítka *Načíst ze souboru* se zvolí soubor s hracím plánem nebo kameny a po potvrzení je možné jej editovat (viz 7.4.1 a 7.4.2). Dalším tlačítkem je *Editor hracího plánu.* Po kliknutí se zobrazí okno, ve kterém se volí velikost plánu (v rozmezí 10 – 26). Po potvrzení této volby se v pravé části zobrazí editovatelný hrací plán (viz 7.4.1). Posledním tlačítkem je *Editor hracích kamenů.* Zde se zobrazí okno s volbou jazykové sady písmen. Poté co je zvolen jeden z jazyků, zobrazí se editor hracích kamenů (viz 7.4.2).

7.4.1 Editace hracího plánu

Tento editor (obr. 7.6) je určen na úpravu a vytváření vlastních hracích plánů. V levé části okna je několik přepínačů. Jejich pomocí se volí jaký druh pole se na plán má umístit. To se poté provede jednoduchým kliknutím na příslušné místo hracího plánu v pravé části okna. Významy polí jsou následující. První čtyři značí pole, která znamenají bonus pro položené kameny a to dvoj- nebo troj-násobek pro jeden kámen nebo celé položené slovo. Další značí pole, na kterém musí první hráč začínat svůj tah. Je tedy nezbytné, aby na plánu bylo alespoň jedno, neboť bez něho by byl každý tah neplatný. Poslední volba značí pole bez jakéhokoliv bonusu, či zvláštního významu.

Po skončení editace se plán uloží pomocí tlačítka *Uložit*.

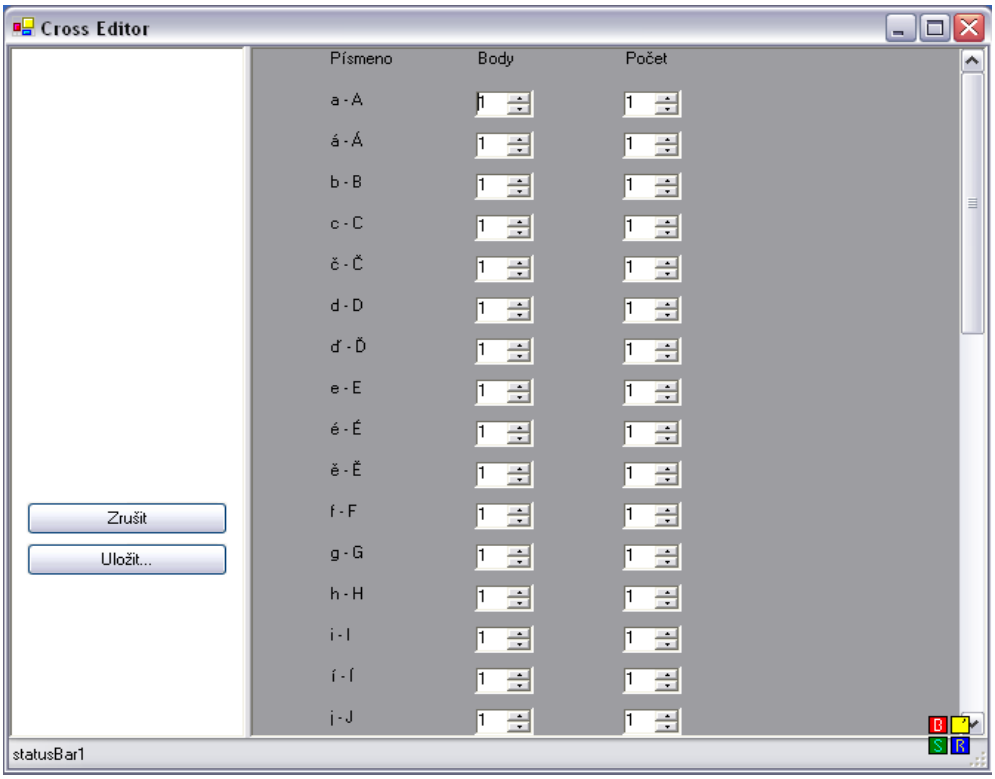


Obrázek 7.6: Editor hracího plánu

7.4.2 Editace hracích kamenů

V této sekci (obr. 7.7) je možné nastavovat složení kamenů v sáčku. Po zobrazení editoru se v pravé části vypíší všechna písmena zvolenho jazyka. U každého písmene můžete nastavit jeho četnost v sáčku a také počet bodů, které za něj hráč po položení obdrží.

Po skončení editace se data uloží pomocí tlačítka *Uložit*.



Obrázek 7.7: Editor hracích kamenů

8 Závěr

Práce byla zpracována za účelem seznámení se s teorií týkající se implementace hry Scrabble s možností hry proti počítači. Tento teoretický základ měl být poté využit při programování aplikace hrající Scrabble. Tato aplikace měla umožňovat i hru po síti.

Prvním cílem mé práce bylo seznámení se s teorií vztahující se k implementaci počítačem řízeného hráče a k uložení slovní zásoby. V teoretické části (kapitoly 3 a 4) uvádím přehled postupů a algoritmů pro efektivní implementaci hry. Pro úsporné uložení slovníku a následné rychlé vyhledávání je v kapitole 3 zmíněna struktura s názvem trie. Tuto strukturu lze efektivně reprezentovat pomocí minimalizovaných deterministických konecných automatů. Dále je pak zmíněn algoritmus pro generování tahů, jehož rychlost výrazně ovlivňuje hratelnost hry. Základem algoritmu je zjednodušení problému pomocí předpočítávaných množin.

Dalším cílem bylo hru naprogramovat tak, aby byla hratelná jak pro začátečníky, tak pro hráče středně zdatné. To se mi myslím povedlo díky tomu, že je umožněna jednoduchá volba obtížnosti počítačového hráče. Tím mají i méně zdatní hráči ze hry proti počítačovému protivníkovi požitek a radost z vítězství. Naopak ani pokročilí hráči, díky celkem obsáhlé slovní zásobě se nebudou při hře nudit. Navíc hra je hratelná i přes síťové propojení, takže si ji mohou užít i ti, kteří k sobě nemají zrovna moc blízko. Další přednost hry vidím ve velké míře modifikovatelnosti a to jak z hlediska vytváření vlastních hracích plánů, tak v možnosti hry s omezeným časem což by mohlo ještě zvýšit míru hratelnosti.

Stále je však mnoho věcí, které je možné na hře zlepšovat. Za prvé je to rozšíření slovní zásoby, neboť použitý slovník, i přes svůj rozsah zdaleka neobsahuje všechna povolená slova. Za druhé je to vylepšení strategického myšlení počítače. Je možné například využít toho, že ke konci hry, když už nejsou v sáčku žádné kameny máme plnou informaci o kamenech protivníka. Možné je také dále rozvíjet uživatelské rozhraní.

Literatura

[DAC98] Daciuk, J. *Incremental Construction of Minimal Acyclic Finite State Automata and Transducers,* 1998

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

[CAS] Česká asociace Scrabble (ČAS), *Internetové stránky ČAS,* 2006, http://www.scrabble.hrejsi.cz

[SED03] Sedgewick, R. *Algoritmy v C,* 2003, SoftPress, 80-86497-56-9

[SEL05] Sells, C. *C# a WinForms,* 2005, Zoner Press, 80-86815-25-0

[CHY84] Chytil, M. *Automaty a Gramatiky,* 1984, SNTL, 04-012-84