VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SCRABBLE PRO MOBILNÍ TELEFONY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ONDŘEJ KANĚČKA
AUTHOR

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SCRABBLE PRO MOBILNÍ TELEFONY

SCRABBLE FOR CELLUAR PHONES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDŘEJ KANĚČKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. RUDOLF SCHÖNECKER
SUPERVISOR

BRNO 2007

Abstrakt

Práce zkoumá možnosti vyhledávacích algoritmů a slovníkových datových struktur na platformách s omezeným výpočetním výkonem a dostupnou pamětí (typicky jde o mobilní telefony) a ukazuje jejich výhody a nevýhody v souvislosti s touto platformou. Konkrétně se zabývá jejich uplatněním ve známé stolní hře SCRABBLE. Používá Appel-Jacobsonův vyhledávací algoritmus na hledání možných tahů. Algoritmus má k dispozici slovník se všemi slovy, který je uložen v tzv. struktuře DAWG, která umožňuje slova rychle vyhledávat a současně zajišťuje kompresi obsažených slov, takže je velikost slovníku v paměti vzhledem k cílové platformě dostatečně malá. Výsledná Java aplikace pro mobilní telefon (MIDP 2.0) přidává grafické rozhraní a ovládání hry a umožňuje tak hraní hry SCRABBLE jak proti umělé inteligenci, tak proti jinému člověku.

Klíčová slova

Scrabble, Univerzální slovníková struktura, USS, DAWG, Appel-Jacobsonův algoritmus, prohledávání stavového prostoru, Umělá inteligence, Herní heuristiky

Abstract

This paper deals with an usage of searching algorithms and lexical data structures for platforms with limited computing power and accessible memory (celluar phones and similar) in SCRABBLE brand crossword game. There are also shown its advantages and disadvantages on this platform. The Appel-Jacobson searching algorithm is used to search possible moves. The lexicon with all words is available for this algorithm. It's stored in Directed Acyclic Word Graph (DAWG), which provides fast data searching and data compression too, so the size of the lexicon is comparatively small considering the target platform. The final Java application for celluar phones (MIDP 2.0) adds graphical interface and game control so it provides playing SCRABBLE brand crossword game against artificial intelligence or against human.

Keywords

Scrabble, Trie, Directed Acyclic Word Graph, DAWG, Appel-Jacobson algorithm, Backtracking games, Artificial intelligence, Game heuristics

Citace

Ondřej Kaněčka: Scrabble pro mobilní telefony, bakalářská práce, Brno, FIT VUT v Brně, 2007

Scrabble pro mobilní telefony

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Rudolfa Schöneckera

. . . . . .
Ondřej Kaněčka
14. května 2007

Poděkování

Děkuji Ing. Rudolfovi Schöneckerovi za odborné vedení bakalářské práce, za poskytování rad a materiálů.

© Ondřej Kaněčka, 2007.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **3**

2 Jiné způsoby implementace **5**
2.1 70. a 80. léta . . . 5
2.2 Použití statistických metod . . . 6

3 Reprezentace slovníku **7**
3.1 Univerzální slovníková struktura . . . 7
3.1.1 Implementace USS . . . 7
3.1.2 Hledání slova . . . 8
3.2 DAWG . . . 8
3.2.1 Z USS do DAWG . . . 10
3.3 Uložení slovníku v souboru . . . 11
3.3.1 Struktura uzlu . . . 11

4 Appel-Jacobsonův algoritmus **12**
4.1 Příprava hrací plochy . . . 12
4.1.1 Redukce problému do jednoho rozměru . . . 12
4.1.2 Cross-Check množiny . . . 12
4.1.3 Anchors - spojovací políčka . . . 13
4.2 Generování samotného tahu . . . 13
4.2.1 Levá část . . . 13
4.2.2 Ošetření výjimek . . . 15
4.2.3 Implementační doporučení . . . 15

5 Výběr tahu **16**
5.1 Použité heuristiky . . . 17
5.2 Testování . . . 17

6 Český Scrabble **19**

7 Implementace **20**
7.1 Prototypové programy . . . 20
7.2 Objektový návrh . . . 20
7.3 Oddělené grafické rozhraní . . . 21
7.4 Omezení cílové platformy . . . 21
7.4.1 Práce se soubory . . . 22
7.4.2 Velikost polí . . . 22

7.4.3 Otevírání souboru 22

8 Závěr **23**

Kapitola 1

Úvod

Doba, kterou právě prožíváme, bývá nazývána dobou informačních technologií. Velkou zásluhu na tom mají osobní počítače, které dnes slouží k práci nebo k zábavě téměř v každé domácnosti. Ještě důležitější uplatnění se pro ně nachází v různých společnostech a institucích. Provoz mnoha firem je na nich dnes přímo závislý a k základnímu vzdělání člověka patří i znalost ovládání osobních počítačů, přesněji aplikací, které jsou na nich vykonávané. K masovému rozšíření osobních počítačů došlo v USA v 80. letech, u nás k tomu došlo o deset let později.

V současné době prožíváme jistou paralelu, tentokrát jde ale o mobilní telefony a podobná přenosná zařízení. Dnes jsou mobilní telefony ještě více rozšířené než osobní počítače. To je způsobeno jejich cenou a také to plyne z jejich určení - jde o věc, u které se předpokládá, že jí bude vlastnit a používat jedna osoba. Nejde tedy o poměr jedno zařízení na jednu domácnost (jako u PC), ale o poměr jedno zařízení na jednu osobu. Trh s mobilními telefony je dnes velice perspektivní, stejně tak jako trh s aplikacemi pro ně určenými.

Tato přenosná zařízení bývají označována jako zařízení s omezeným výpočetním výkonem. Je to proto, že jsou porovnávána s dnešními osobními počítači, které představují pro většinu lidí výpočetně nejsilnější stroje - různé servery pro vědecké výpočty apod. nejsou zařízení používáná masově mezi lidmi. Svým výpočetním výkonem a dostupnou pamětí se mobilní telefony vrací zpět do 80. let. Technologicky ale nijak nezaostávájí za osobními počítači, jejich omezení plynou z platformy, kterou představují a u které současně určují vývoj. U této platformy jde především o minimalizaci hardwarových prostředků, tj. snaha zmenšovat fyzické rozměry jednotlivých hardwarových komponent a současně zmenšovat jejich počet. I zde však platí, že výpočetní výkon neustále roste. Možnosti mobilních telefonů jsou dále rozšiřovány používáním operačních systémů (Symbian, MS Windows Mobile Phone Edition apod.). Výpočetní výkon prvních osobních počítačů byl znatelně menší než dnes, ale i tyto stroje byly už schopny provádět matematicky náročné operace. Pro mobilní telefony a podobná zařízení dnešní doby platí v podstatě to samé. K dispozici je množina aritmetických, logických, bitových a relačních operací. Mikroprocesor mobilního telefonu umožňuje programátorovi uživatelských aplikací to, co CPU osobního počítače. Z tohoto pohledu se tedy nemusíme nijak omezovat cílovou platformou a můžeme vytvářet aplikace, které sekvenčně provádějí mnoho různých matematických operací, i když doba jejich provádění je několikanásobně větší než u dnešních PC.

Hlavním cílem projektu bylo vytvořit hru SCRABBLE pro mobilní telefon, která umožňuje hraní proti umělé inteligeci. V tomto případě představuje časově náročné velké množství sekvenčně prováděných matematických operací prohledávání stavového prostoru, které je třeba provádět kvůli generování tahů umělé inteligence. Z uvedených vlastností aplikace je možné odvodit dílčí cíle práce, kterých bylo třeba dosáhnout. Jde především o tři hlavní problémy:

- Najít vhodný algoritmus na vyhledávaní možných tahů.
- Najít vhodnou heuristiku na výběr z nalezených tahů.
- Najít vhodnou reprezentaci slovníku, ve kterém jsou uložena všechna validní slova.

Konkrétní řešení těchto problémů je popsáno v kapitolách 3, 4 a 5.

Kapitola 2

Jiné způsoby implementace

Scrabble je typ hry s neúplnou informací - neznáme písmena, která mají naši soupeři v zásobníku. Pouze na konci hry, když už v sáčku nejsou žádné kameny, můžeme toto odvodit a teoreticky tedy použít algoritmus pro hry s úplnou informací (např. MinMax). Ve vyhledávacím algoritmu výsledné aplikace se toto řešení nevyskytuje. Pracuje se pouze s algoritmy s neúplnou informací.

Dále jsou popsány různé formy implementace hry Scrabble. Jedná se o programy napsané v sedmdesátých a osmdesátých letech, po nich následuje popis jednoho zcela odlišného způsobu řešení výběru tahu.

2.1 70. a 80. léta

Programy, které hrají Scrabble a samy generují tahy se začaly psát s rozvojem samotné disciplíny umělé inteligence. Jistý zvrat ve vývoji znamenal Appel-Jacobsonův algoritmus (viz 3) v kombinaci se slovníkem USS či DAWG (viz 4). Většina dnešních aplikací [8] používá tento algoritmus. V sedmdesátých a osmdesátých letech vzniklo několik více či méně úspěšných programů na hraní Scrabblu, které nevyužívají Appel-Jacobsonův algoritmus a které stojí za zmínku:

MONTY Komerční produkt vhodný pro množství počítačů a zařízení té doby. Používá strategický i taktický koncept hraní. Expertní hráči ho trvale poráželi.

Peter Turcan napsal hráče Scrabblu, který prohledává slova ve slovníku podle jejich délky a začíná nejdelšími. Pro každý tah se rozhoduje zda a kde by měl být umístěn na hrací ploše. Jeho program neusiluje a o nějaký nepříjemný tah pro soupeře ale používá ohodnocující funkci, která je více důmyslná než jen výběr nejlépe ohodnoceného tahu. Bere v úvahu aktuální skóre hráčů, aktuální stav hrací plochy a kameny zbylé v zásobníku.

Peter Weinberger napsal generátor tahů, který nejdříve ohodnotí každé políčko na hrací ploše - uloží informaci o písmenech a pozicích, které vyhovují (podobně jako u Appel-Jacobsona). Pak jsou uloženy další informace o zásobníku a hrací ploše jako celku. Potom se podívá na každé slovo ve slovníku a rychlými globálními testy každé nevyhovující slovo vyřadí. Zbylá slova jsou podrobena rychlím lokálním testům (pro každé políčko) a vyřadí ty, která nelze umístit na jednotlivé pozice. Písmena každého zbylého slova jsou zkoušena na každé pozici a zjišťuje se, zda tvoří validní tahy. Program nemá žádný strategický koncept - jednoduše vybírá nejlépe ohodnocený tah.

Stuart Shapiro napsal několik programů hrajících Scrabble v jazycích SIMULA a Pascal. Používají slovníky uložené jako strom, kde každý uzel reprezentuje písmeno (3.1). Průchod stromem od

kořene generuje slovo, které je tvořeno z písmen procházených uzlů. Písmena se při průchodu objevují přibližně v pořadí podle hodnoty písmene ve Scrabblu (sestupně). Důvod, proč se výše hodnocená písmena dávají blíže ke kořeni stromu je, aby se nejhodnotnější slova našla dříve(v případě částečného průchodu slovníkem). Generátor tahů nezkouší všechny možné pozice na hrací ploše, pouze pozice předem posouzené. Jakmile je tah shledán vyhovujícím (přinejmenším podle minimálního skóre), je vybrán.

U popsaných algoritmů jsou doby generování tahů různé. Záleží na velikosti jejich slovníků. Na současných strojích jsou to řádově sekundy až desítky sekund. Appel-Jacobsonův algoritmus se výrazně odlišuje právě rychlostí a v porovnání s nimi je mnohonásobně rychlejší. Záleží ale na konkrétním tahu. Pokud má hráč k dispozici žolík nebo dokonce oba dva, pak se velikost prohledávaného prostoru může enormně zvětšit a najít nejlepší tah netrvá desetiny sekund, ale desítky sekund.

2.2 Použití statistických metod

Za zmínku stojí řešení [4], kdy je pro výběr tahu na začátku a ve střední části hry použit algoritmus pro hry s úplnou informací v kombinaci s metodou Monte Carlo. Algoritmus je v tomto případě několikrát proveden a jeho různé výsledky jsou statisticky vyhodnoceny. Podle tohoto statistického výsledku lze zjistit tah, který bude pravděpodobně nejvýhodnější.

Podstata metody Monte Carlo ale spočívá v mnohonásobném provedení daného algoritmu. Kolikrát se algoritmus provede, závisí na požadované přesnosti. Z toho lze tušit, že toto řešení není vhodné pro naše účely, kde nám jde především o rychlost nalezení nejvýhodnějšího tahu.

Kapitola 3

Reprezentace slovníku

Během generování tahu počítačem se prohledáváním do hloubky hledají všechny možné kombinace písmen v zásobníku a na hrací ploše, které tvoří slova uložená ve slovníku. Vzhledem ke způsobu prohledávání a k tomu, že je kladen velký důraz na rychlost prohledávání a v neposlední řadě i na velikost slovníku, je reprezentace slovníku např. jako abecedně seřazený seznam slov nevhodná. Vhodná struktura pro tyto účely je tzv. univerzální slovníková struktura (USS, Trie). Tuto strukturu lze dále převést na tzv. strukturu DAWG (Directed Acyclic Word Graph), která poskytuje srovnatelnou rychlost hledání slov, ale velikost takto uloženého slovníku může být i několikrát menší než USS.

3.1 Univerzální slovníková struktura

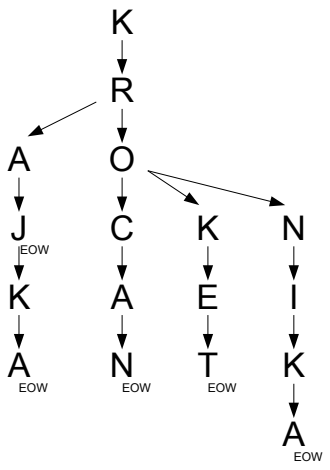
Jde o druh stromu, který díky svým vlastnostem automaticky zajišťuje datovou kompresi v paměti a neomezuje přístup k uloženým informacím. Viz [10]. Základní myšlenka USS vychází z následujícího zjištění: součástí mnoha slov jsou stejné kořeny a slova se různí pouze svými předponami a příponami. Vezměme například skupinu slov: KROCAN, KROKET, KRAJKA, KRONIKA, KRAJ. Pokud je uložíme ve tvaru stromu podle obrázku 3.1, pak místo původních 29 znaků jich uložíme 17. V případě obsáhlejšího slovníku bude zisk ještě větší. Předpokládáme, že ve slovníku uložíme ve velké míře série slov začínajících stejnými písmeny - např. úplné skloňování podstatných jmen atd.

3.1.1 Implementace USS

Implementace se poněkud liší od běžných způsobů ukládání slov. Neukládají se zde totiž slova nebo jejich části a přesto je účel splněn. V jazyce C může deklarace struktury USS vypadat např takto:

```
typedef struct slovnik
{
  struct slovnik *t[N+1];
} USS;
```

Jedná se o deklaraci rekurzivního typu, jehož jediným prvkem je pole ukazatelů na hodnoty stejného typu. Znaku “A” odpovídá prvek t[0], znaku “B” zase t[1] atd. Konstanta N značí počet rozlišovaných písmen. Poslední prvek s indexem N indikuje koncový uzel - to je uzel, ve kterém končí nějaké slovo. Hlavní princip USS je objasněn na příkladu znázorněném na obrázku 3.2.



Obrázek 3.1: Komprese dat v USS

Abychom mohli provést názorný rozbor, omezíme počet znaků abecedy na 6: A, K, L, O, P, S. Písmeno "A" představuje první ukazatel v každém uzlu. Písmeno "S" je potom předposlední ukazatel. Poslední ukazatel značí konec slova (EOW - End Of Word). Hodnota o konci slova nabývá jen dvou stavů ANO/NE (končí/nekončí), proto je v obrázku označována jako *TRUE* / *FALSE*. Každý uzel obsahuje pole t o rozměru 7. Pokud je poslední ukazatel roven hodnotě *TRUE*, znamená to, že v tomto místě dané slovo končí. Které slovo to je lze zjistit průchodem stromu od kořene. Z kořenového uzlu se lze dostat do jakéhokoliv jiného uzlu. Chceme se tedy dostat z kořenového uzlu do konkrétního koncového uzlu. Během průchodu si zaznamenáváme ukazatele, které procházíme (tedy písmena, která reprezentují). Až se dostaneme do koncového uzlu, ukončíme průchod stromem a písmena hledaného slova máme zaznamenána.

V příkladu na obrázku 3.2 jsou tedy uložena tyto slova: KOLA, KOLAPS, KOLO, LOS, SAKO.

3.1.2 Hledání slova

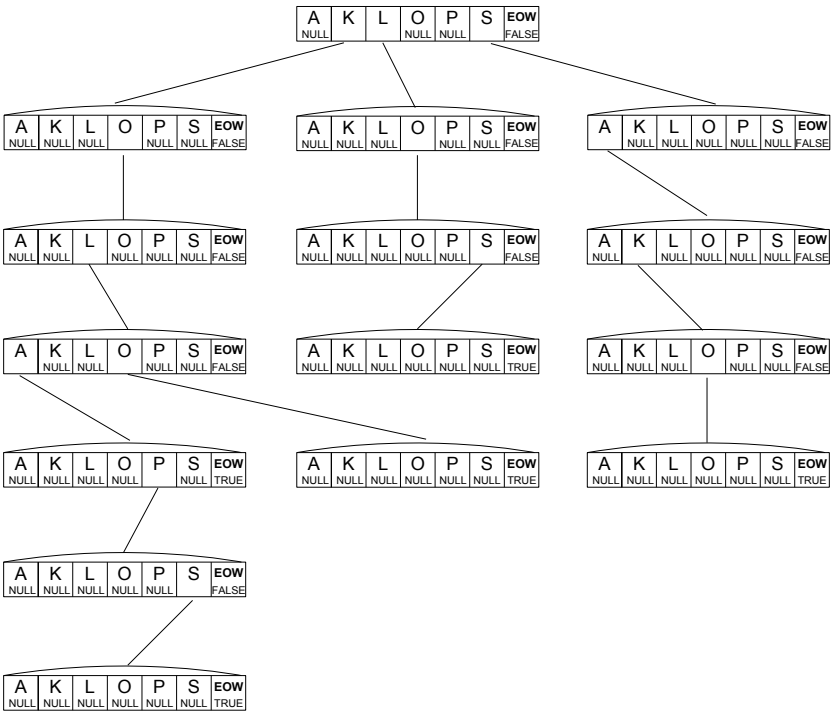
Pokud hledáme nějaké konkrétní slovo, pak procházíme strom od kořene a v každém uzlu čteme hodnotu příslušného ukazatele, která značí adresu synovského uzlu. Index příslušející k aktuálně procházenému písmenu určuje, ze kterého ukazatele máme adresu číst. (V kořenovém uzlu je aktuální písmeno první písmeno slova.) Během průchodu stromem pak mohou nastat tyto dvě situace:

1. Dostaneme se až ke koncovému uzlu - hledané slovo je ve slovníku.

2. Narazíme na ukazatele s hodnotou *NULL* a nemůžeme tedy dále procházet strom. Hledané slovo proto není ve slovníku.

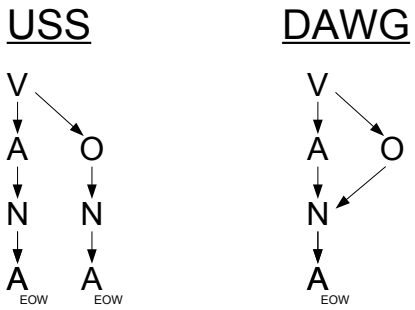
3.2 DAWG

Struktura DAWG je podobná USS. Také zde existují uzly, jejichž průchodem hledáme slova ve struktuře obsažená. Algoritmus hledání slova je stejný a pokud zůstane stejná i struktura uzlů, pak lze zcela stejný algoritmus použít na hledání slov v DAWG a současně v USS. Rozdíl mezi DAWG



Obrázek 3.2: Zápis řetězců v USS

a USS je v tom, že USS tvoří strom, zatímco DAWG tvoří graf, podobný grafům konečných automatů. Na obrázku 3.3 je vidět rozdíl uložení slov dvou slov VANA a VONA. Rozdíl tedy spočívá v tom, že zatímco v USS je komprese dat založena na totožnosti začátků slov, u struktury DAWG vzniká komprese dat i díky stejným písmenům na konci slova. Z obrázku 3.3 je vidět, že počet uzlů ve struktuře DAWG je menší, než v USS. V tomto případě je dokonce nejmenší možný. Lze totiž vytvořit mnoho DAWG struktur nad určitou stejnou množinou slov, ale pouze jedna má minimální počet uzlů. Z toho také plyne hlavní význam DAWG: Zatímco rychlost vyhledávání slov v USS a DAWG je v podstatě stejná, tak velikost slovníků DAWG bývá menší, než u USS. Závisí na typu uložených slov, jaký je poměr mezi velikostmi těchto dvou struktur (obsahují-li stejnou množinu slov). V případě velké podobnosti slov může být DAWG několikanásobně menší než USS.



Obrázek 3.3: Rozdíl mezi DAWG a USS

3.2.1 Z USS do DAWG

Zajisté existuje mnoho způsobů, jak vytvořit DAWG strukturu. Následuje popis vytvoření slovníku DAWG s minimálním počtem uzlů pomocí již vytvořeného slovníku USS dle [5].

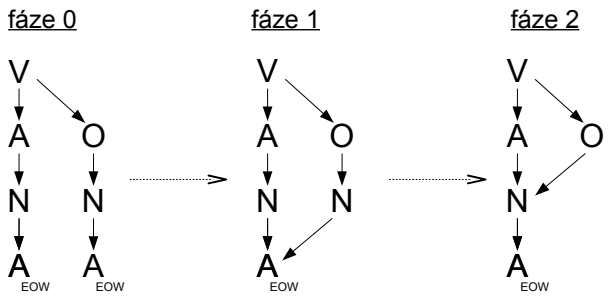
Před začátkem konverze z USS na DAWG je třeba si označit některé údaje u každého uzlu v USS:

- **Maska ukazatelů** - Informace o každém ukazateli, zda někam ukazuje nebo má hodnotu *NULL*.
- **Zanoření** - Počet přechodů z kořenového uzlu do daného uzlu. Kořen má zanoření 0.
- **Hloubka potomka** - Nejvyšší počet uzlů, přes které projdete z daného uzlu do listového uzlu. Listy mají hloubku potomka 0.

Hlavní důvod označování uzlů je, že podle toho lze rychle zjistit, zda jsou uzly identické. Podle masky ukazatelů a hloubky potomka je možné toto určit jednoznačně. Proč to tak je bude zřejmé dále. Poté, co si označíme uzly, si ještě vytvoříme pomocná pole uzlů. V každém takovém poli budou uzly (jejich odkazy/reference) se stejnou hloubkou potomka.

Nyní máme uzly označené a seřazené. Začneme s uzly s hloubkou potomka 0. Jestliže uzly X a y jsou identické (všechny uzly s hloubkou potomka 0 jsou identické, protože mají stejnou masku ukazatelů a stejnou hloubku potomka 0), potom každý uzel, který obsahuje ukazatel (má potomka) na uzel Y přesměruji (přepíši ukazatel) na uzel X. Když hledáme uzly, které by mohly obsahovat ukazatele na uzel X nebo Y, pak stačí tyto uzly hledat v poli s uzly s hloubkou potomka 1 a nemusíme procházet celou strukturou USS. Ještě nebylo vysvětleno to, proč přesměrujeme ukazatele na uzel X a ne na Y. Obecně bychom si mohli vybrat mezi uzly náhodně, ale v tomto projektu je slovník uložen v souboru a platí zde proto určitá pravidla, která jsou objasněna v kapitole o uložení slovníku v souboru (3.3) a která také využívají údaj zanoření uzlu.

Pokud se ještě vrátíme k příkladu z obrázku 3.3, pak by struktura DAWG po této první fázi, kdy už všechny uzly s hloubkou potomka 1 ukazují na jediný uzel s hloubkou potomka 0 (list), vypadala jako fáze 1 znázorněná na obrázku 3.4. Analogicky se poté provádí to samé s uzly s hloubkou potomka 1. Teď již víme, proč stačí k určení identičnosti dvou uzlů pouze údaje o masce ukazatelů a o hloubce potomka. Je to proto, že pokud jsou tyto dva údaje stejné, pak víme, že všechny ukazatele těchto dvou uzlů ukazují na identickou množinu uzlů (ukazatel uzlu A s indexem i ukazuje na stejný uzel, jako ukazatel uzlu B se stejným indexem). A ukazují na identickou množinu uzlů proto, protože tyto uzly byly už v předešlých fázích konverze sjednoceny. Všechny fáze konverze USS na DAWG ukazuje obrázek 3.4.



Obrázek 3.4: Fáze tvorby DAWG z USS

3.3 Uložení slovníku v souboru

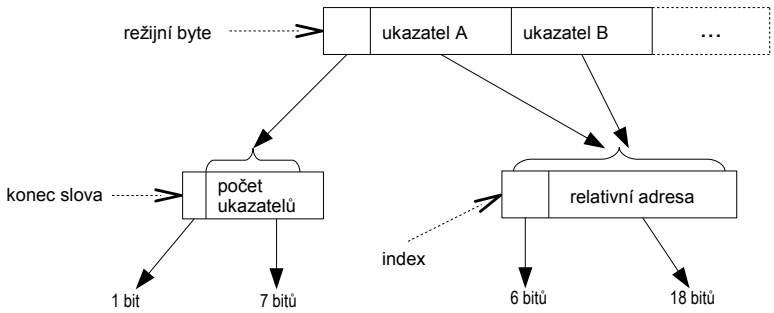
Slovník představuje perzistentní data. Aplikace, která se slovníkem pracuje, ho musí nejdříve načíst ze souboru nebo může pracovat přímo se souborem. Slovník je tvořen množinou uzlů (DAWG či USS), které jsou uloženy za sebou v souboru. První je uložen kořenový uzel, následují uzly se zanořením 1 atd. Hodnoty ukazatelů uzlů jsou kladná celá čísla, která odkazují na adresu v souboru, kde je uložen počáteční byte odkazovaného uzlu. Aby byla tato čísla co nejmenší, neukládá se do nich absolutní adresa uzlu (pozice od počátku souboru) ale relativní adresa uzlu vůči uzlu, ze kterého čteme hodnotu ukazatele (o kolik bajtů se musí poskočit z aktuální pozice v souboru). To, že je odkazovaný uzel vždy uložen na vyšší adrese v souboru než uzel, který na něj ukazuje, je v případě DAWG struktury dáno tím, že se přesměrovávají ukazatele vždy na uzel s vyšším zanořením ve struktuře USS, zatímco uzel s menším zanořením se z DAWG struktury vyloučí. Viz 3.2.1. V případě USS se toto řešit nemusí, protože ukazatele konkrétního uzlu zde vždy ukazují na uzel se zanořením o 1 větším.

3.3.1 Struktura uzlu

Velký podíl na velikosti slovníku má kromě přirozené komprese, vznikající díky podobnosti slov, také struktura samotného uzlu.

Uzel nese informace o hodnotách ukazatelů. Ukazatelů je ale poměrně velký počet (v českém Scrabblu jich je celkem 40). Mnoho těchto ukazatelů ale vůbec nikam neukazuje - slovo zde končí. Nejvíce ukazatelů je nastavených v uzlech s menším zanořením a postupně přibývá těch, které mají hodnotu *NULL*. Poslední v řadě jsou listy, které nemají ani jednoho ukazatele nastaveného. Záleží na podobnosti slov, ale průměrný počet nastavených ukazatelů v uzlu je 2 až 5 (ze 40). Proto existuje hodně zbytečných ukazatelů, které nikam neodkazují a zabírají zbytečně místo ve slovníku.

Toho využívá způsob uložení uzlu, používaný v projektu. Nenastavené ukazatele uchovávají totiž jenom informaci o tom, že nikam neukazují. Tato informace nemusí zabírat prostor velikosti ukazatele ale jen jednoho bitu. Proto se před samotné ukazatele vkládá na počátek uzlu jeden režijní byte, který nese informaci o počtu nastavených ukazatelů (0-39) a informaci o konci slova (ANO/NE). Po přečtení režijního bytu tedy víme, kolik dalších bytů uložených bezprostředně za tímto režijním bytem patří k danému uzlu. Ukazatel má velikost 3 byty a jsou v něm uchovávány dvě informace: hodnota samotného ukazatele (18 bitů) a index (písmeno, které představuje) ukazatele (6 bitů). Ukazatele jsou v uzlu uloženy vzestupně (podle jejich indexů) a toho využíváme při binárním vyhledávání ukazatele s konkrétním indexem. Obrázek 3.5 ukazuje strukturu uzlu názorně.



Obrázek 3.5: Struktura uzlu uloženého v souboru

Kapitola 4

Appel-Jacobsonův algoritmus

V této kapitole jsou popsány kroky přípravy hrací plochy pro aplikaci Appel-Jacobsonova algoritmu a algoritmus samotný. Následuje popis algoritmu převzatý z [1]. K dispozici je stále původní program [9] psaný v jazyce C pro operační systém UNIX.

4.1 Příprava hrací plochy

Aby bylo vyhledávání slov co nejrychlejší, je třeba před samotným generováním tahu zjistit některé údaje o hrací ploše. Zrychlení a hlavně zjednodušení celého algoritmu spočívá také v redukci problému do jednoho rozměru.

4.1.1 Redukce problému do jednoho rozměru

Slova ve Scrabblu se mohou tvořit buď zprava doleva nebo shora dolů. Můžeme se omezit jen na tvoření slov zprava doleva tím, že si vytvoříme druhou hrací plochu, která k ní bude transponovaná. Algoritmus tedy nejprve bude vyhledávat slova na první a potom na druhé ploše (na pořadí nezáleží). Tyto herní plochy se musí samozřejmě po každém tahu synchronizovat. To znamená, že písmeno vložené na první hrací plochu na souřadnice x,y se vloží i na transponovanou hrací plochu, ale na souřadnice y,x.

4.1.2 Cross-Check množiny

Když vložíme nové slovo (omezili jsme se jen na zprava doleva), pak nově vložená písmena musí navazovat i na slova ve vertikálním směru, která jsou na ploše již umístěná. Před každým tahem tedy musíme znát políčka, která mohou tvořit i slova ve vertikálním směru. Jsou to políčka umístěná přímo pod, nebo nad nějakým již zaplněným políčkem. Jinými slovy u každého políčka na hrací ploše musíme znát množinu písmen (Cross-Check množinu), které lze na toto políčko vložit, aby písmena tvořily i vertikální slova.

Tato činnost je časově náročná, mnohdy trvá déle, než samotné generování slova. Chceme-li čas strávený výpočtem Cross-Check množin eliminovat, musíme si uvědomit následující. Před prvním tahem nemusíme tyto množiny počítat - z pohledu Cross-Check množin lze na každé políčko vložit jakékoliv písmeno. Po vložení slova s pozicí prvního písmene na x,y a s délkou slova d můžeme omezit počet políček na hrací ploše, na kterých se nově počítají Cross-Check množiny. Jsou to políčka na x-ových souřadnicích od x do $x+d-1$. Y-ové pozice existují pak maximálně dvě pro každou x-ovou pozici. Jsou to první prázdná políčka, ke kterým se dostaneme průchodem po hrací ploše nahoru nebo dolu z každého nově vloženého políčka. Pokud je slovo vloženo na herní plochu

vertikálně, pak se situace zjednodušuje pouze na jednu x-ovou souřadnici a složitější část se zase musí provádět nad transponovanou hrací plochou.

4.1.3 Anchors - spojovací políčka

Anchor políčka jsou políčka, která spojují část slova již vloženého na plochu a část slova nově vytvořeného. Přesná definice zní: "Nejlevější nově vložené políčko připojené k políčku, které je již zaplněné." Jedno vkládané slovo má tedy jedno anchor políčko/písmeno a na jednom řádku může být několik anchor políček. Anchor políčka jsou tedy všechna prázdná políčka sousedící (zprava, zleva, shora, zdola) s již zaplněným políčkem. Jako u Cross-Check množin je třeba před každým tahem tyto políčka znát.

4.2 Generování samotného tahu

Pro každý řádek hrací plochy se problém redukuje na následující jednorozměrný problém:

1. Daná písmena, která lze použít.
2. Obsah aktuálního řádku na hrací ploše.
3. Cross-Checky a anchor políčka na aktuálním řádku.
4. Vygenerovat všechna možná slova na aktuálním řádku.

Pro každé anchor políčko na každém řádku hrací plochy (i na transponované) je třeba provést samotné hledání slov, které se skládá ze dvou částí:

1. Najít všechny možné levé části slova. (Levá část je nalevo od anchor políčka.)
2. Ke každé levé části slova najít všechny správné pravé části slova. (Pravá část je napravo od anchor políčka, včetně anchor políčka.)

Jde tedy o prohledávání do hloubky (backtracking).

4.2.1 Levá část

O levé části slova platí:

1. Její začátek (nejlevější políčko) je buď na konci hrací plochy nebo před jiným anchor políčkem. (Délka levé části může tedy být i nulová.)
2. Je tvořena buď z nově vložených políček nebo z již zaplněných políček, ale nikdy ne dohromady. (To plyne z definice anchor políčka a z prvního bodu.) Pokud levou část tvoří již vložená políčka, pak se tato část slova již nemusí generovat, jednoduše stačí použít písmena na daných políčkách.
3. Její políčka jsou jednoduchá Cross-Check políčka. To znamená, že na políčko lze vložit jakékoliv písmeno, které máme samozřejmě v daném tahu ve svém zásobníku. (To také plyne z definice anchor políčka a z prvního bodu.)

U levé části lze tedy určit její maximální délku (viz bod 1). Funkce, která generuje levou část slova je v pseudokódu zapsána takto: (Funkce pracuje se slovníkem, který je reprezentován strukturou USS nebo DAWG.)

```
LeváČást(ČástSlova, uzel U, limit)
{
  RozšiřujDoprava(ČástSlova, U, AnchorPolíčko)
  jestliže limit > 0
  {
    pro každý platný přechod P z uzlu U
    {
      jestliže písmeno p, které reprezentuje přechod P máme v zásobníku
      {
        vyjmi písmeno p ze zásobníku
        nechť U' je uzel dosažený z přechodu P
        LeváČást(ČástSlova+p, U', limit-1)
        vlož písmeno p zpět do zásobníku
      }
    }
  }
}
```

Nechť k je maximální délka levé části slova. Tuto funkci voláme pro každé anchor políčko hrací plochy takto: *LeváČást(“”, kořenový uzel, k)*.

Uvažujme funkci *KorektníTah(slovo)*, které předáváme validní slovo a tato funkce jej ukládá. Jednoduchá funkce si může pamatovat jen nejlépe ohodnocený tah. Funkce generující pravé části slova je zapsána takto:

```
RozšiřujDoprava(ČástSlova, uzel U, políčko Q)
{
  jestliže políčko Q je prázdné
  {
    jestliže U je konečný uzel
    {
      KorektníTah(ČástSlova)
    }
    pro každý platný přechod P z uzlu U
    {
      jestliže písmeno p, které reprezentuje přechod P máme v zásobníku
      a jestliže písmeno p se nachází v Cross-Check množině daného políčka
      {
        vyjmi písmeno p ze zásobníku
        nechť U' je uzel dosažený z přechodu P
        nechť Q' je políčko napravo od políčka Q
        RozšiřujDoprava(ČástSlova+p, U', Q')
        vlož písmeno p zpět do zásobníku
      }
    }
  }
  jinak
  {
    nechť p je písmeno na políčku Q
```

```
      jestliže U má platný přechod, který je reprezentován písmenem p
      a tento přechod směřuje k uzlu U'
      {
        nechť Q' je políčko napravo od políčka Q
        RozšiřujDoprava(ČástSlova+p, U', Q')
      }
    }
  }
```

4.2.2 Ošetření výjimek

Existují některé výjimky, které je třeba v kódu ošetřit:

- **Prázdné prefixy -** Pokud voláme funkci LeváČást a políčko nalevo od anchor políčka je už zaplněné, pak levou část negenerujeme, ale použijeme písmena vložená nalevo od anchor políčka. To znamená, že voláme přímo funkci RozšiřujDoprava s prvním parametrem, který obsahuje písmena z levé části (musíme zachovat pořadí písmen). Druhý parametr je uzel, ke kterému se dostaneme průchodem písmen levé části od kořene. Poslední je dané anchor políčko.

- **Žolíky -** Tyto kameny reprezentují kterékoliv písmeno. Vždy, když nenajdeme ve svém zásobníku požadované písmeno, tak se ještě podíváme, zda nemáme žolík. Pokud ano, pak ho necháme reprezentovat hledané písmeno. Když pak žolík vkládáme zpět do zásobníku, tak mu vracíme jeho mnohotvárnost.

4.2.3 Implementační doporučení

Je dobré zvětšit šířku hrací plochy o jedno políčko doprava. Políčka napravo budou mít prázdnou Cross-Check množinu (žádné písmeno nelze vložit) a současně budou prázdná (žádné písmeno na ně není vložené). Tyto políčka slouží jako zarážky při hledání pravých částí slov.

Při počítání Cross-Check množin je také výhodné pro každé prázdné políčko současně počítat bodové ohodnocení vertikálních slov, které mohou při tahu vzniknout. Je to součet hodnot všech políček tvořících sled políček s políčky přímo nad a pod daným prázdným políčkem.

Kapitola 5

Výběr tahu

Úkolem Appel-Jacobsonova algoritmu je nalézt všechny validní tahy z dostupných písmen a z aktuálního stavu hrací plochy. Úkolem herních heuristik je vybrat jeden z těchto tahů a vložit jej na hrací plochu. To znamená, že ve hře je uplatněn i jistý strategický prvek, který hodnotí tah nejen podle jeho bodového ohodnocení, ale i podle jeho umístění na hrací ploše. Heuristika, která jednodušše vybírá nejlépe ohodnocený tah se nazývá hladová strategie. V této práci je význam herních heuristik dvojí. První je nalézt takové heuristiky, aby se u nich projevila strategická schopnost umisťovat tahy na herní plochu tak, že jsou schopny porážet hladovou strategii s více jak 50% pravděpodobností. Druhý význam je v podstatě opačný. Jde o heuristiky, které úmyslně vybírají tahy s menším bodovým ohodnocením a v souboji s hladovou strategií jsou většinou poraženy. Jde o způsob simulování hráče s menší slovní zásobou. Oba typy heuristik mají společné to, že nevybírají vždy tahy s nejvyšším bodovým ohodnocením. Obě to ale dělají z jiného důvodu. První se tím snaží porazit ostatní heuristiky, druhá se jimi snaží být poražena.

I když hladová strategie představuje nejjednodušší způsob, jak vybrat tah, jde současně o velice účinnou strategii. Všechny ostatní heuristiky, které ji mají porážet, totiž mohou vybírat tahy se stejným nebo menším bodovým ziskem. Lze nalézt způsob, jak ji výrazně porážet [4]. Tento způsob je ale založen na statistickém vyhodnocování dat, která se získávají několikanásobným během patřičných algoritmů a je tedy časově náročný a nevhodný pro použití v této práci. Vhodnější jsou heuristiky, které se přirozeně snaží bránit soupeři vkládat výhodné tahy na hrací plochu. Způsobů, jak toho docílit, je mnoho a jsou různě účinné. Mohou brát v úvahu stav hrací plochy, stav zásobníku, vlastní a soupeřovo skóre, fázi hry (začátek, konec), apod.

Jak konkrétně by se měla taková heuristika chovat, se dá pochopit z některých vybraných tipů (které jsou v tomto projektu využity a které lze najít v manuálu pro hru), jak hrát správně Scrabble:

- "Písmena X, Ď, Ň, Ť jsou nejlépe hodnocená písmena v sáčku. Neschovávejte si je, hrajte je pokud možno co nejdříve. Tato písmena mají svou vysokou bodovou hodnotu právě pro svou obtížnou umístitelnost. Na druhou by bylo škoda nevytěžit z nich maximum. Pokuste se najít nejlepší umístění ve slově s prémií."

- "Nejlepším kamenem je žolík. Výrazně zvyšuje možnost vytvoření tahu ze všech sedmi kamenů a tak získání padesátibodové prémie. Neužívejte však žolíka příliš lehkovážně. Je lepší uchovat si jej do chvíle, kdy budete moci vytvořit tah s minimálně 30bodovým ziskem."

- "Tahy s trojnásobnou slovní prémií jsou ve hře velmi důležité a mnohdy rozhodují o celkovém výsledku. Takže neusnadňujte svému soupeři možnost jejich realizace."

- "Hráč, který hraje jen občas, nerad mění kameny s odůvodněním, že nechce ztrácet zbytečně

tah. Pokud ale opakovaně hrajete tahy s bodovým ziskem 5 bodů či méně z důvodu málo vhodných kamenů, pak je výměna pravděpodobně nejlepším řešením."

- "Na konci hry, kdy se skóre již pomalu uzavírá, je často velmi důležité dohrát jako první. Vašemu soupeři zůstanou v zásobníku kameny, jejichž hodnotu odečte od svého bodového zisku a vám naopak tyto body skóre vylepší."

5.1 Použité heuristiky

Ve hře bylo použito několik heuristik. Jejich různé kombinace byly testovány, aby se nalezla taková celková heuristika, která předčí hladovou strategii.

- **Vkládání žolíků** - Je nevýhodné vytvářet tahy, ve kterých jsou použity žolíky a které mají současně malé bodové ohodnocení. Proto pokud má být slovo obsahující žolík nebo žolíky vloženo na hrací plochu, musí mít určité minimální bodové ohodnocení. Podle doporučení, jak hrát správně Scrabble (viz 5) je toto minimum 30 bodů. Tato heuristika může výrazně zvýšit skóre hráče, který vlastní žolíka. Na druhou stranu žolíky ve hře existují pouze dva a jejich ovlivnění hry je proto velice náhodné (to je ale jejich účel) - různí hráči dostanou během hry rozdílný počet žolíků.

- **Minimální bodový zisk tahu** - V případě, že hráč vkládá slovo, které má bodový zisk menší, než nějaká hranice, může se rozhodnout slovo nevložit a raději zvolí výměnu zásobníku. Používaná heuristika mění celý zásobník, aby byl po výměně v zásobníku co nejvíce vyrovnaný počet souhlásek a samohlásek. Právě jejich nevyrovnanost je totiž nejčastější důvod, proč nelze vytvořit delší slovo s vyšším bodovým ziskem. Minimální bodová hranice, kdy se zásobník ještě nevyměňuje, byla předmětem testování (viz 5.2).

- **Bezpečný tah** - Bezpečný tah je takový tah, který neumožní soupeři na tento tah navázat a obsadit např. políčko s trojnásobnou slovní hodnotou. Takový tah má většinou menší bodovou hodnotu, než bodově nejlepší tah v daném tahu hráče. Znamená to, že místo tahu s největším bodovým ziskem se heuristika rozhodne položit jiný tah s méně body, ale má jistotu, že neumožní soupeři vytvořit bonusový tah (viz další bod). Z nalezených bezpečných tahů se snaží umístit vždy ten bodově nejlepší.

- **Bonusový tah** - Bonusový tah má alespoň jedno písmeno vloženo na bonusovém políčku, např. opět na trojnásobné slovní prémii. Hráč s touto heuristikou se přednostně snaží pokládat tahy na bonusová políčka (vybírá nejvýše hodnocený tah) a dává jim tak přednost před bodově nejlépe hodnocenými tahy. Zároveň tak ale znemožňuje soupeři položit slovo na toto bonusové políčko.

- **Nejdelší tah** - Hráč se snaží vytvářet co nejdelší slova. Většinou neplatí, že nejdelší slovo je zároveň slovo s nejvyšším bodovým ziskem. Tato heuristika je používána v závěrečné fázi hry, když už je v sáčku méně jak sedm hracích kamenů a hráči tedy už nemohou měnit své zásobníky. Místo toho se hráč snaží položit co nejdelší slovo, aby tak mohl dříve uzavřít hru, nebo alespoň aby mu bylo po skončení hry odečteno méně bodů z jeho celkového skóre.

5.2 Testování

Testováním heuristik a jejich kombinací se hledala nejlepší heuristika, která by s co největší pravděpodobností porážela hladovou strategii. Aby byly výsledky testování objektivní, hráli proti sobě

v jedné hře vždy jen dva hráči. Náhodným prvkem ve hře je tahání písmen ze sáčku, proto nebylo možné dostat přesný výsledek pouze z jedné hry mezi stejnými soupeři. Vždy se hrálo alespoň 150 her. Výhodu ve hře má ten hráč, který pokládá první slovo. Proto měli oba hráči stejný počet prvních tahů.

Při testování se zjistilo, že výše popsané heuristiky bezpečný tah, bonusový tah a nejdelší tah pokládají většinou tahy s výrazně menším bodovým ziskem, než hladová strategie. Takové heuristiky byly ještě méně úspěšné, než hladová strategie. Proto bylo třeba nastavit jistou maximální bodovou hranici rozdílu mezi vybraným a nejlépe hodnoceným tahem. Když je tato hranice překročena, chová se heuristika jako hladová strategie a pokládá nejlépe hodnocený tah. To se děje přibližně v polovině tahů. Heuristiky, které takto vybírají tahy, jsou schopny (když je hranice rozdílu správně nastavena) s větší jak 50% pravděpodobností porazit hladovou strategii. V tabulce 5.1 jsou zobrazeny výsledky testů, kdy proti sobě vždy hrály vybraná heuristika a hladová strategie. První sloupec ukazuje nastavení heuristiky. Hodnoty v pořadí značí:

1. Maximální bodový rozdíl nejdelšího a bodově nejlepšího tahu.
2. Minimální bodová hodnota slova, které se může položit na hrací plochu. Při menší hodnotě už se vyměňuje zásobník.
3. Maximální bodový rozdíl bezpečného a bodově nejlepšího tahu.
4. Maximální bodový rozdíl bonusového a bodově nejlepšího tahu

Z výsledků je patrné, že nejlépe nastavené heuristiky dokáží porážet hladovou strategii s asi 56% pravděpodobností. Jiné heuristiky jsou vyrovnané a některé jsou naopak hladovou strategií ve většině případů poražené. To je způsobeno tím, že mají nastavené velké hodnoty některých nebo i všech parametrů. Vybírají tedy s větší pravděpodobností tahy, které jsou sice strategičtěji umístěné na herní plochu, ale mají o příliš mnoho bodů méně, než maximálně bodově ziskový tah. Čím vyšší hodnoty parametrů jsou nastaveny, tím menší obtížnost hráče-počítače.

Parametry	Počet her	Výhry		Remízy		Prohry	
		počet	%	počet	%	počet	%
-1 3 1 6	150	82	54,6	4	2,6	64	42,6
1 6 1 6	150	78	52	0	0	72	48
1 3 1 10	150	58	38,6	0	0	92	61,3
1 5 2 5	150	80	53,3	2	2,3	68	45,3
-1 0 2 5	150	79	52,6	1	1,3	70	46,6
2 6 1 6	150	84	56	1	1,3	65	42,6
2 10 1 6	150	75	50	0	0	75	50

Tabulka 5.1: Hladová strategie proti různým heuristikám

Kapitola 6

Český Scrabble

V porovnání s anglickým jazykem je čeština jazyk velmi ohebný. Slova mají mnoho různých tvarů. Skloňují se, časují se, mají různé předpony a přípony. Navíc má čeština více písmen. Tyto její vlastnosti s sebou přináší různé nevýhody, které bylo třeba řešit v projektu.

Pokud bychom chtěli, aby náš program generoval všechna přípustná slova podle pravidel Scrabblu [7], pak bychom museli mít všechna tyto slova ve všech tvarech uložena ve slovníku. Problém je v enormní velikosti takového slovníku. Slovník s českými slovy pouze v základním tvaru má přibližně 58 000 slov. Rozšířený slovník se všemi tvary těchto slov jich má pak asi 1 200 000. I když budou slova uložena ve struktuře DAWG, kde se uplatní podobnost slov (komprese slov vzroste, protože se zvětší počet podobných slov), bude velikost tohoto slovníku stále několikanásobně větší. To např. celkově zpomalí generování tahů, ale hlavně se tu projeví nevýhody cílové platformy (7.4). Tento problém má několik (bohužel jenom částečných) řešení:

1. Lze použít tzv. lematizátory. Jsou to programy, které dokáží slovo z nějakého tvaru převést na tvar základní. Využívají přitom vzorů slov a podobně. V tomto směru ale v češtině existuje mnoho výjimek a ne u všech slov se dá použít jejich vzor k nalezení správného tvaru. To činí lematizátory opět do jisté míry závislé na velkých slovnících.

2. Pokud validujeme slovo, které není v základním tvaru a které tedy nemáme uložené ve slovníku, můžeme validovat pouze část slova (několik počátečních písmen). V tomto případě tedy nelze zjistit, zda je celé slovo opravdu validní či nikoliv.

3. Slovník může obsahovat slova ve všech tvarech jen do určité délky slova. Slova delší než je jistá hranice jsou uložena jen v základním tvaru.

Poslední řešení je také použito v tomto projektu. Slovník obsahuje všechny tvary maximálně pětipísmenných slov. To je důležité protože většina slov umístěných na hrací ploše mají délku do pěti písmen. Těchto slov je asi 28 500. Celkem je pak ve slovníku uloženo 83 000 slov a velikost DAWG slovníku obsahující tuto množinu slov je asi 330 kB.

Aby nebyl hráč-člověk omezen množinou uložených slov, lze ve výsledné aplikaci validaci slov vypnout a slovník je pak využíván pouze při generování tahu počítačem.

Kapitola 7

Implementace

Výsledná aplikace se snaží simulovat všechny prvky obsažené ve hře. Oproti stolní hře přidává možnost hrát proti umělé inteligenci (nebo nechat hrát UI proti sobě). Grafické rozhraní tvoří vlastní kreslicí plocha i standardní formulářové prvky dostupné v MIDP 2.0. Ovládání hry je intuitivní a je popsáno v nápovědě aplikace. Pro ukládání dat, tak aby byla přístupná i po vypnutí a znovu spuštění aplikace, slouží tzv. RMS (Record Management System), česky systém pro správu záznamů. Ve hře lze takto ukládat nastavení jednotlivých hráčů.

7.1 Prototypové programy

Bylo třeba vytvořit dva prototypové programy (konzolové aplikace v jazyce C). První sloužil k vytvoření samotného slovníku DAWG ze seznamu slov (uložených v obyčejné textové podobě). Dále se na něm testovala vhodná struktura uzlu slovníku DAWG. Druhý program byl určen k testování heuristik pro výběr tahu. Obsahoval tedy Appel-Jacobsonův algoritmus na generování tahů včetně všech nezbytných funkcí pro ohodnocení tahu, pro obsluhu zásobníku (doplňování, měnění obsahu), pro vkládání tahu na hrací plochu a pro celkový běh programu (hráči se střídali v jednotlivých tazích a na konci hry se zapsaly výsledky).

7.2 Objektový návrh

Všechny fyzické objekty, které se objevují ve skutečné hře, jsou implementovány jako objekty i v aplikaci. Jde o herní plochu, zásobník s kameny, sáček s kameny, slovník a hráče. Aplikace definuje ještě další třídy, následuje popis všech tříd.

- **GameBoard** - Udržuje informace o stavu hrací plochy. Obsahuje dvojrozměrné pole prvků (15 x 15), které uchovávají různé informace o každém políčku. Obsahuje metody pro čtení a zapisování informací do těchto políček. Nejdůležitější z nich je nastavování Cross-Check množin a jejich čtení.

- **Rack (zásobník s kameny)** - Udržuje informace o stavu hráčova zásobníku. Protože při generování tahů počítačem se se zásobníkem neustále manipuluje (vkládají a vyjímají se písmena), je obsah reprezentován jako pole integerů o velikosti 40 (počet písmen). Prvek pole značí četnost daného písmene v zásobníku. Kromě vkládání a vyjímání písmen obsahuje zásobník další metody jako doplnění zásobníku písmeny ze sáčku, výměnu všech nebo některých písmen.

- **Bag (sáček s kameny)** - Obsahuje metody pro vyjímání a vkládání písmen. Ze sáčku se vždy vyjme náhodné písmeno - generuje se náhodné číslo s rozsahem jako počet písmen zbylých v sáčku. Po vyjmutí se část obsahu sáčku posouvá tak, aby při dalším vyjímání stačilo generovat pouze jedno náhodné číslo, které značí vybrané písmeno (jeho pozici v sáčku).
- **DAWG (slovník)** - Představuje rozhraní pro práci se slovníkem. Pracuje přímo se soubory, ve kterých jsou slovníky uloženy. Nejdůležitější je metoda pro čtení obsahu uzlů. Ta se používá jak při generování tahů počítačem, tak při validaci slova vloženého člověkem. Obsah slovníku je neměnný, neexistují metody pro zapisování.
- **Player** - Obsahuje informace o nastavení hráče a o tazích, které hrál.
- **ScrabbleMidlet** - Potomek třídy Midlet. Obsahuje všechny screeny (seznamy, formuláře a varovná hlášení), která jdou ve hře zobrazit. Dále obsahuje výkonný kód všech příkazů, která náleží k jednotlivým screenům.
- **GameService** - Obsahuje většinou statické metody, se kterými pracují instance všech ostatních tříd.
- **BoardCanvas (kreslicí plátno)** - Potomek třídy Canvas. Obstarává vykreslování hrací plochy a zásobníku s kameny (ty jsou stále na obrazovce). Obsahuje kód, který je prováděn při stisku jednotlivých kláves na mobilním telefonu.
- **Move** - Reprezentuje tah umístěný hráčem na hrací plochu. Ukládá informace o rozmístění nově vložených kamenů. Obsahuje metody pro validaci slova vloženého člověkem a pro vkládání tahu na hrací plochu.
- **MoveGenerator** - Implementuje Appel-Jacobsonův algoritmus na generování tahů. Dědí třídu Thread, protože generování tahů běží jako samostatné vlákno. Je to proto, aby bylo možné během tahu počítače ovládat hru.

7.3 Oddělené grafické rozhraní

Z předchozí kapitoly je zřejmé, že na grafickém rozhraní se podílí jedna třída a vykreslování na obrazovku je tedy zcela nezávislé na použitých algoritmech. To je běžný způsob implementace využívající objektově orientovaného programování. Takový návrh umožňuje snadnější údržbu a aktualizace různých částí aplikace. Některé komponenty ale není tak snadné od sebe oddělit. Například algoritmus pro generování tahů pracuje se slovníkem DAWG - předpokládá tedy určitou strukturu slovníku.

7.4 Omezení cílové platformy

Cílová aplikace je psána v rozšíření Javy J2ME (Java 2 MicroEdition), které vyhovuje standardu MIDP 2.0 [6]. Tento standard je podporovaný zejména mobilními telefony, pagery, palmtopy a podobnými zařízeními. To s sebou obecně přináší jistá omezení oproti osobním počítačům. Jde především o menší velikost a rychlost operační paměti. Výpočetní výkon je také několikanásobně menší. Omezení spojená s velikostí prostoru pro ukládání souborů však nebyla při této práci nijak na obtíž.

7.4.1 Práce se soubory

Ve standardu MIDP 2.0 nejsou přítomny jisté třídy pro práci se souborem (např. **RandomAccessFile**). Jsou zde pouze třídy pro tzv. nízkoúrovňovou práci se souborem. Tyto třídy neumožňují jednoduchou změnu pozice v souboru na určité místo nebo dotazování na aktuální pozici v souboru. Jsou tu pouze metody pro označení aktuální pozice a nastavení pozice na tuto značku. Tento způsob pohybu kurzoru po souboru je ale zcela nedostačující pro práci se slovníkem DAWG nebo USS.

Proto je třeba část nebo celý slovník uložit do proměnné v programu a s tou pak neomezeně pracovat.

7.4.2 Velikost polí

V předchozí části bylo řečeno, že je třeba slovník zkopírovat do proměnné v programu. Typicky jde o pole bytů. Paměť, kterou může program alokovat má ale také jistou maximální velikost. Záleží na typu zařízení, jaká tato velikost je.

Aby byla aplikace spustitelná na většině zařízení, musí se velikost pole, do kterého se ukládá slovník, nějak omezit. Lze to udělat tak, že se slovník rozdělí na více menších slovníků. Efektivní je, když jsou slovníky rozděleny podle délky slov. Rozdělíme slovník např. na dva menší. V prvním jsou slova s maximální délkou např. 6, ve druhém pak zbytek. Při generování tahu se načte nejprve první slovník, a po jeho průchodu pak druhý. Během jednoho generování tahu počítačem resp. při validaci slova vloženého člověkem se slovník načítá ze souboru právě jednou resp. nejvýše jednou. Čas strávený načítáním slovníku do proměnné je ve srovnání s generováním tahu téměř zanedbatelný.

7.4.3 Otevírání souboru

Při otevírání souboru, který obsahuje slovník, došlo k výjimce. Ta byla způsobena nedostatkem paměti. Soubor se proto musel rozdělit na několik částí a ty se pak zvlášť otevíraly a postupně ukládaly do proměnné v programu.

Kapitola 8

Závěr

Hlavním cílem této práce bylo vytvořit hru Scrabble pro mobilní telefon, která se snaží simulovat hraní stejnojmenné stolní hry. Parametry hry odpovídají oficiálním pravidlům Scrabblu [7]. Jde např. o počet hráčů, správné rozměry hrací plochy, rozmístění bonusových polí na hrací ploše, počet písmen ve hře, jejich četnosti a bodové ohodnocení. Dále pak jde o způsoby vkládání slov na hrací plochu včetně jejich správného bodového ohodnocení. Je zahrnuta možnost výměny písmen v zásobníku i možnost vynechání tahu. Aplikace se svým vzhledem a možnostmi snaží eliminovat některá omezení plynoucí z cílové platformy, na které je implementována. To znamená, že se snaží neomezovat hráče a umožňuje mu udržovat si přehled o herní ploše a jeho zásobníku s písmeny.

Úkolem hráče je vkládat na hrací plochu smysluplná slova tak, aby měl na konci hry větší skóre, než jeho soupeři. Po vložení slova aplikace sama slovo validuje (rozhodne o tom, zda nově položená písmena tvoří slovo či ne). Ve hře to znamená, že písmena buď umožní nebo neumožní vložit na herní plochu. Hráčem může být člověk a nebo počítač (umělá inteligence). To znamená, že aplikace je schopna tahy vytvářet, validovat a z vytvořených validních tahů jeden vybrat a vložit jej na hrací plochu.

Prvním krokem bylo najít vhodný slovník - tedy vhodnou množinu slov, vůči které by program porovnával zadaná slova a testoval jejich validitu. Ohledy se musely brát především na velikost výsledného slovníku a to jak z důvodu omezení paměti, tak kvůli tomu, že s velikostí slovníku roste prohledávaný prostor a tedy i doba potřebná k nalezení slova. Po určení vhodného rozsahu slovníku bylo třeba najít vhodnou reprezentaci (formát) slovníku a slovník samotný vytvořit. Hra generuje tahy, to znamená, že prohledává stavový prostor, který je tvořen slovníkem. Chceme-li tah vygenerovat za rozumnou dobu, pak odezva slovníku musí být co nejmenší. Přesněji řečeno jeho odpověď, zda dané slovo existuje. Výsledná aplikace je ale spouštěna v prostředí s omezenými zdroji (paměť, výkon). Mobilní telefony a různá přenosná zařízení se různí svými prostředky. Pokud chceme, aby byla hra spustitelná na většině těchto zařízení, je třeba se přizpůsobit. Velikost slovníku by se tedy měla pohybovat v řádech kilobajtů. Je ale pravda, že tato přenosná zařízení se zdokonalují a jejich výpočetní výkon a dostupná paměť roste. Proto by mohl mít náš slovník velikost mnohonásobně větší a aplikace by stále byla spustitelná na většině těchto zařízení.

Slovník tedy musí být jednoduše rychlý a malý. Obě tyto podmínky splňuje tzv. Univerzální slovníková struktura. Ještě lepší kompresní vlastnosti poskytuje tzv. struktura DAWG (Directed Acyclic Word Graphs), která je použita i v tomto projektu.

Struktura DAWG poskytuje relativně rychlý a malý slovník, vhodný pro použití v této práci. Existuje však další slovníková struktura podobná struktuře DAWG nazývaná GADDAG [3]. I tato struktura má podobu konečného automatu. na rozdíl od DAWG je toto dvousměrný graf. To znamená, že každý uzel kromě dopředných ukazatelů obsahuje navíc zpětné ukazatele na všechny uzly, které zase obsahují dopředný ukazatel na tento uzel. Tím vzniká dvousměrná vazba podobně

jako např. v lineárně vázaném dvousměrném seznamu. Kvůli zpětným ukazatelům je GADDAG větší než DAWG (typický slovník s anglickými slovy je pětkrát větší), ale rychlost vyhledávání se může až zdvojnásobit. Rychlost se zvýší díky možnosti procházet slovník v opačném směru. Tento druh prohledávání je specifický pro hru Scrabble, protože slova navazují na písmena, která jsou již umístěna na hrací ploše. Generování tahů by se díky tomuto řešení tedy výrazně zrychlilo, přičemž velikost slovníků by byla stále přijatelná i na platformách používaných pro mobilní telefony a podobná přenosná zařízení. Tímto směrem lze tedy dále pokračovat v této práci.

Máme-li slovník, musíme napsat algoritmy, které s ním pracují. Forma vyhledávacích algoritmů je přímo závislá na struktuře slovníku, se kterým pracují. Jde o jednosměrnou závislost vyhledávacích algoritmů na slovníku protože stavový prostor, který algoritmus prohledává, je tvořen právě slovníkem. Naproti tomu slovník není nijak závislý na vyhledávacích algoritmech. Požadavky kladené na tyto algoritmy jsou podobné jako požadavky na slovník. Algoritmus musí být rychlý (toho docílíme maximálním ořezáním hledaného prostoru) a nesmí být náročný na paměť. Algoritmus používaný v této práci se nazývá Appel-Jacobsonův a díky tomu, že ořezává prohledávaný stavový prostor maximální měrou, splňuje uvedené podmínky velice dobře.

Nejjednodušší a mnohdy i nejefektivnější je vložit v daném tahu na herní plán slovo, které je ze všech nalezených slov nejlépe ohodnocené (hladová strategie). Tento způsob výběru tahu je většinou pro běžné hráče téměř neporazitelný. Jedním z cílů této práce je testovat různé herní heuristiky, které vybírají tah nejen podle jeho bodového ohodnocení, ale i podle jeho umístění na herní ploše nebo podle fáze hry (začátek, konec). Tyto heuristiky tedy do jisté míry napodobují logické uvažování hráče-člověka. Měly by být schopny hrát vyrovnané partie s hladovou strategií a dokonce ji předčit. Aby bylo hodnocení heuristik spravedlivé, musí samozřejmě všechny pracovat se stejným slovníkem.

Rozšiřování práce lze soustředit na hledání nových a vylepšení stávajících heuristik pro výběr tahu. Aby byla úspěšnost heuristik proti hladové strategii co nejvyšší, je také potřeba je vyladit - tedy najít k nim správné parametry. Ty lze najít tak, že různé kombinace heuristik s různě nastavenými parametry necháme hrát proti sobě. Tím jsme schopni nalézt jednu vítěznou heuristiku. Jde tedy o princip tzv. genetického algoritmu. Podle [2] lze dosáhnou až 63% úspěšnosti s hladovou strategií. V této práci bylo dosaženo pouze 56% pravděpodobnosti (viz 5.2).

Ve výsledné aplikaci je možnost zvolit obtížnost hráče-počítače. Jde o napodobovaní chování hráče s menší slovní zásobou. To znamená, že je použita heuristika s opačným účelem než předchozí heuristiky - jejím úkolem je vybírat tahy záměrně s menším bodovým ohodnocením tak, aby na konci hry byl bodový zisk hráče používajícího tuto heuristikou menší než u ostatních hráčů.

Literatura

[1] Andrew W. Appel and Guy J. Jacobson. The world's fastest scrabble program. `http://www.gtoal.com/wordgames/jacobson+appel/aj.pdf`, 1988. Dostupná v květnu 2007.

[2] Filip Dvořák. Nastavení heuristik. `http://ksvi.mff.cuni.cz/~mraz/nn/nnpres07/Scrabble.pdf`. Dostupná v květnu 2007.

[3] Steven A. Gordon. A faster scrabble move generation algorithm. `http://www.cs.ubc.ca/rr/proceedings/spe91-95/spe/vol24/issue2/spe880.pdf`, 1994. Dostupná v květnu 2007.

[4] Jan Pomikálek. Monte carlo metoda ve hře scrabble. `http://nlp.fi.muni.cz/~xpomikal/scrabble/`, 2004. Dostupná v květnu 2007.

[5] WWW stránky. Directed acyclic word graphs. `http://www.wutka.com/dawg.html`. Dostupná v květnu 2007.

[6] WWW stránky. Mid profile. `http://java.sun.com/javame/reference/apis/jsr118/`. Dostupná v květnu 2007.

[7] WWW stránky. Oficiální pravidla scrabble. `http://scrabble.hrejsi.cz/cgi/cas/index.pl?volba=pravidla/index.htm`. Dostupná v květnu 2007.

[8] WWW stránky. Scrabble - source code. `http://www.gtoal.com/wordgames/scrabble.html`. Dostupná v květnu 2007.

[9] WWW stránky. Appel-jacobson original form. `http://www.gtoal.com/wordgames/gatekeeper/crab.sh.txt`, 1992. Dostupná v květnu 2007.

[10] Piotr Wróblewski. *Algoritmy, datové struktury a programovací techniky*. Computer Press, 2004. ISBN 80-251-0343-9.

Příloha

Pravidla Scrabblu

Jde o stolní hru, kde spolu mohou hrát až čtyři hráči. Ti se snaží vkládat na hrací plochu písmena, která mají každý zvlášť ve svém zásobníku. Všechna písmena na hrací ploše musí po každém tahu tvořit skutečná slova a to buď zprava doleva nebo shora dolů. Každé vložené slovo (kromě prvního) musí navazovat na nějaké již vložené písmeno. Hru tvoří tři základní prvky:

- sáček se všemi kameny
- zásobník s kameny (každý hráč má k dispozici jeden)
- hrací plocha, na kterou se kameny vkládají

Jak začít hru

Všechny hrací kameny jsou vloženy do sáčku. Na začátku hry si každý hráč vytáhne ze sáčku jeden hrací kámen. Hráč, který si vytáhne kámen s písmenem nejbližším začátku abecedy, začíná. Vytažené kameny se vrátí do sáčku.

Každý hráč si pak vylosuje sedm nových hracích kamenů a uloží si je do zásobníku. Začínající hráč provede první tah a pak táhnou postupně další hráči. Každý hráč má ve svém tahu na výběr: umístění slova na hrací plochu nebo výměnu kamenů nebo vynechání tahu ("PASS").

Výměna kamenů

Každý hráč může využít svého tahu k výměně některých či všech svých kamenů. Nejdříve vyjme vybrané kameny ze zásobníku, pak vylosuje stejný počet kamenů ze sáčku, umístí je do zásobníku a nakonec zbylé kameny vrátí do sáčku. Podmínkou výměny je minimálně sedm hracích kamenů v sáčku, a to i v případě, že hráč chce měnit jen jeden hrací kámen.

"PASS" (vynechání tahu)

Namísto tvoření slov na herním plánu nebo výměny kamenů může hráč ohlásit PASS - tj. vynechat tah. To může udělat vždy, podle svého uvážení, ať je či není schopen z písmen ve svém zásobníku utvořit nová slova.

Ale pokud všichni hráči ohlásí PASS ve dvou po sobě následujících kolech, hra končí.

Umístění prvního slova

První hráč zkombinuje dva či více kamenů, vytvoří slovo a položí je na herní plán, aby slovo bylo čitelné zleva doprava či shora dolů a jeden z kamenů ležel na středovém poli. Slova položená diagonálně jsou neplatná.

Všechny kameny hrané v jednom tahu musí být položeny pouze v jedné linii, a to vodorovně nebo svisle.

Přípustná slova

U běžné hry hráči rozhodují o správnosti slova podle vybraných slovníků, na kterých se všichni shodnou. V této práci je k dispozici jeden slovník s 83 000 slov a hráči-počítače podle něj tvoří slova. Hráč-člověk si může vypnou validaci jeho slov a nemusí hrát podle tohoto slovníku.

Hrací plán

Hrací plán tvoří 15 x 15 políček, na která se vkládají hrací kameny s písmeny. Mezi normálními políčky existují také prémiová pole s různými bonusy pro skórování.

Písmenné prémie znásobují bodové hodnocení písmen na ně položených. Jsou dva druhy: dvojnásobné a trojnásobné písmenné prémie.

Slovní prémie znásobují bodovou hodnotu celých slov na ně položených. Opět existují dva druhy: dvojnásobné a trojnásobné slovní prémie.

Pokud je slovo vloženo současně na písmennou i slovní prémii, vypočte se nejprve hodnota slova se započtením písmenných prémií a slovní prémie pak násobí celou tuto hodnotu slova.

Pokud je v jednom tahu slovo položeno na více slovních prémií současně, pak se tyto prémie násobí.

Bonus prémiového pole se započítává pouze v tahu, kdy je na toto pole položen hrací kámen.

Hrací kameny

Hra obsahuje 100 hracích kamenů, 98 kamenů s jednotlivými písmeny abecedy a dva kameny představující žolíky (mohou představovat jakékoliv písmeno).

Každý hrací kámen má svou bodovou hodnotu. Žolík nemá žádnou bodovou hodnotu.

Počítání bodů

Bodový zisk tahu je součtem hodnot jednotlivých kamenů v nově vytvořených slovech v daném tahu včetně bonusů vyplívajících z umístění kamene na prémiové pole.

Hráč, který v jednom tahu užije všech sedmi svých kamenů, obdrží prémii 50 bodů. Tato prémie se připočte k bodům získaným v tomto tahu až po započtení všech prémií.

Ukončení hry

Hra končí, pokud jsou všechny kameny rozebrány mezi hráči a jeden z hráčů už nemá ani jeden kámen. Konec nestane také tehdy, pokud všichni hráči ve dvou po sobě jdoucích kolech vyhlásí “PASS”.

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

Ovládání programu

Vytváření hráčů

Po prvním spuštění hry je třeba vytvořit nějaké hráče, aby bylo s kým hrát. U hráče lze nastavit tyto tři parametry:

1. **Jméno** - slouží pro identifikaci hráče
2. **Ovládání** - hráč je ovládán buď člověkem nebo umělou inteligencí
3. **Obtížnost** - v případě ovládání hráče UI lze nastavit 5 úrovní obtížnosti

Když je vytvořen alespoň jeden hráč, lze spustit samotnou hru.

Postup hraní hry

Na začátku hry se provádí výběr hráčů, kteří se jí zúčastní. Je nutno vybrat alespoň jednoho a nejvýše čtyři hráče (podle pravidel). Cílem hry je tvořit slova z písmen v zásobníku a vkládat je na herní plochu. Zásobník a herní plocha jsou dva nejdůležitější prvky a jsou proto neustále zobrazeny, aby o nich měl hráč v průběhu celé hry přehled.

Hráč-člověk vloží na hrací plochu písmena a potvrdí je. Tím se spustí validace nově vloženého slova. V případě validního tahu se slovo vloží na herní plochu a pokračuje se ve hře. V opačném případě se hra může dostat do dvou stavů:

1. Slovo je správně vloženo na herní plochu (podle pravidel), ale nevyskytuje se ve slovníku. Tento stav je hráči indikován tak, že nově vložené kameny s písmeny změní svou barvu na červenou. Hráč potom musí svůj tah upravit a pokusit se o nový.
2. Slovo není na herní plochu vloženo podle pravidel. Takové slovo se i přesto může vyskytovat ve slovníku, ale pravidla Scrabblu tah neumožňují vložit. Jsou to tahy, kdy např. slovo nenavazuje na již vložené písmeno, slovo není umístěno v jednom řádku či sloupci, atd. V tomto případě se zobrazí upozornění vysvětlující neuznání tahu a hráč musí zvolit tah jiný.

Během tahu počítače (kdy prohledává stavový prostor slovníku) lze neustále sledovat hrací plochu a pohybovat se po ní. Hráč-člověk se při tomto stavu dostane do pozice pozorovatele a je mu znemožněno jakkoliv zasahovat do hry (kromě ukončení hry).

Na každý tah je vymezen určitý časový interval (pro člověka i UI). Pokud do této doby hráč nezahraje validní tah nebo nevymění písmena v zásobníku, automaticky se jeho tah bere jako “PASS”.

Obrázek 8.1: Ukázka ze hry