VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NÁSTROJ PRO ANALÝZU PSANÍ UŽIVATELE NA KLÁVESNICI

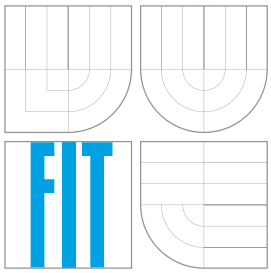
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAROSLAV MOLTAŠ

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NÁSTROJ PRO ANALÝZU PSANÍ UŽIVATELE NA KLÁVESNICI

A TOOL FOR ANALYSIS OF USER'S TYPING SKILLS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAROSLAV MOLTAŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. ADAM HEROUT, Ph.D.

BRNO 2012

Abstrakt

Tato diplomová práce se zabývá analýzou psaní uživatele na klávesnici v systému Windows. Je zde popsána technika hmatové metody sloužící pro rychlé psaní na klávesnici. Dále jsou v práci rozvedeny možnosti zachytávání kláves a tvorby grafického uživatelského rozhraní ve Windows. Součástí zprávy je také popis segmentace napsaného textu a techniky vyhodnocování úrovně psaní. Další část rozebírá implementaci systému. Výsledná aplikace shromažďuje data o psaní uživatele na klávesnici a vyhodnocuje kvalitu psaní a chybovost.

Abstract

This master's thesis deals with analysis of user's typing skills in Windows system. Touch method technique for fast keyboard typing is described. The possibilities of Windows keyboard hooking and GUI making are mentioned. The description of segmentation of written text and writing level evaluation techniques are part of this thesis. Another part deals with the system implementation. The result application collects data about user's typing skills and evaluates the quality of typing and number of mistakes.

Klíčová slova

analýza psaní na klávesnici, hmatová metoda, keylogging, segmentace textu, GUI, Win Forms, grafy

Keywords

typing skills analysis, touch method, keylogging, text segmentation, GUI, Win Forms, charts

Citace

Jaroslav Moltaš: Nástroj pro analýzu psaní uživatele na klávesnici, diplomová práce, Brno, FIT VUT v Brně, 2012

Nástroj pro analýzu psaní uživatele na klávesnici

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Adama Herouta, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jaroslav Moltaš
21. května 2012

Poděkování

Rád bych poděkoval vedoucímu práce, doc. Ing. Adamovi Heroutovi, Ph.D., za jeho vstřícný a přátelský přístup a za pomoc při tvorbě této práce ve formě konzultací.

Obsah

Kapitola 1

Úvod

Psaní na klávesnici je v dnešní době součástí života téměř každého člověka. Je náplní každodenní práce v různých oborech a často prvním kontaktem s počítačem. Proto je snahou psaní na klávesnici urychlit a udělat je komfortnější. Jsou vyvinuty různé techniky a způsoby psaní. Každý uživatel je jedinečný a schopnosti ovládání klávesnice se značně liší.

V této diplomové práci se zabývám analýzou psaní uživatele na klávesnici v systému Windows. K získání dovedností naučit se psát všemi deseti bylo vyvinuto několik více či méně úspěšných programů. Mezi jeden z nejznámějších můžeme zařadit např. All Ten Fingers[1]. Výsledkem této práce ale není aplikace, která by měla uživatele naučit všemi deseti psát, nýbrž tyto schopnosti zdokonalit. Aplikace snímá stisky kláves a z těchto informací vyhodnocuje údaje, které by měly uživateli ukázat statistiky jeho psacích schopností a zároveň nedostatky, kterých se při psaní dopouští, aby mohl na svých dovednostech zapracovat a z chyb se poučit.

Diplomová práce je členěna do několika kapitol. Po úvodu následuje kapitola popisující historii psaní všemi deseti prsty a techniku hmatové metody sloužící pro rychlé psaní na klávesnici.

Ve třetí kapitole se zmíním o keyloggingu, tedy akci, při které jsou zachycovány klávesy. Dále popíšu komunikaci aplikací v systému Windows a to, jaké jsou možnosti zachycování kláves v tomto systému a jak fungují. Především se bude jednat o hákování klávesnice.

Čtvrtá kapitola rozebírá návrh vyhodnocování informací o napsaném textu. Popisuji zde způsoby segmentace napsaného textu. Dále se zde zabývám technikami, které určují kvalitu psaní. Jedná se zejména o popis algoritmu pro detekci chyb a dalších věcí, které jsou aplikací vyhodnocovány, jako např. rytmičnost psaní a počet úhozů.

V páté kapitole se zaobírám implementací systému a jeho jednotlivých částí, převážně popisem algoritmů popsaných v předchozí kapitole. Dále je zde obeznámen princip zachycování kláves a vytvoření grafického uživatelského rozhraní.

Poslední kapitola hodnotí výsledky práce a naznačuje cesty, kterými by mohlo probíhat vylepšení a další rozvoj systému.

Do diplomové práce byly převzaty některé části ze semestrálního projektu. Jedná se především o kapitoly popisující hmatovou metodu (2) a keylogging (3). Kapitoly popisující návrh vyhodnocení informací o psaní (4) a implementaci (5) byly pozměněny a doplněny.

[1] http://www.vsemideseti.cz/

Kapitola 2

Rychlé psaní na klávesnici

V této kapitole se budu věnovat teorii psaní na klávesnici. Blíže se zaměřím na historii a nejrozšířenější způsob psaní všemi deseti – hmatovou metodu.

2.1 Historie

Prvotní nápad psacího stroje pochází z roku 1868 od filozofa Christophera Sholese, který pracoval na přístroji, jehož účelem mělo být automatické číslování stránek knih. Jeden jeho kolega však dostal nápad rozšířit zařízení tak, aby tisklo celou abecedu.

První psací stroj se nazýval „Sholes & Glidden Type Write“ a byl konstruován výrobcem zbraní E. Remingtonem (proto někdy označován také jako „Remington No. 1“) v letech 1874-1878. Druhá část názvu pochází od mechanika Carlose Gliddena, který Sholesovi při výrobě pomáhal. Stroj neměl příliš velký úspěch (méně než 5000 prodaných kusů), ale stal se základem pro světový průmysl.

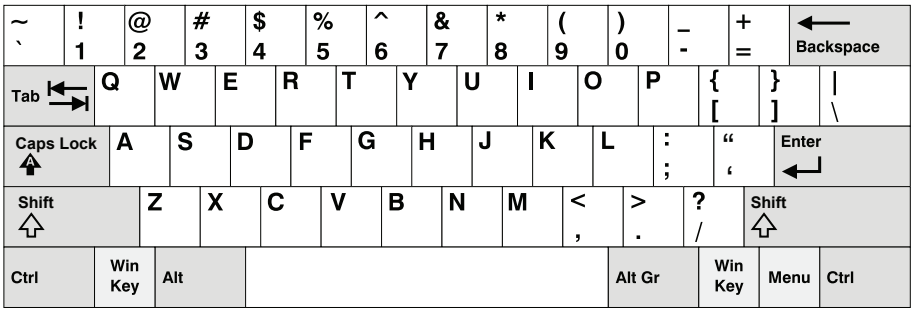
Tento stroj uměl psát pouze majuskule[1] a byl označován jako „blind writer“, tzn., že písař neviděl na napsaný text. Vylepšený model „Remington No. 2“ byl představen v roce 1878 a umožňoval psát majuskule i minuskule[2] pomocí klávesy Shift [7].

Rozložení klávesnice

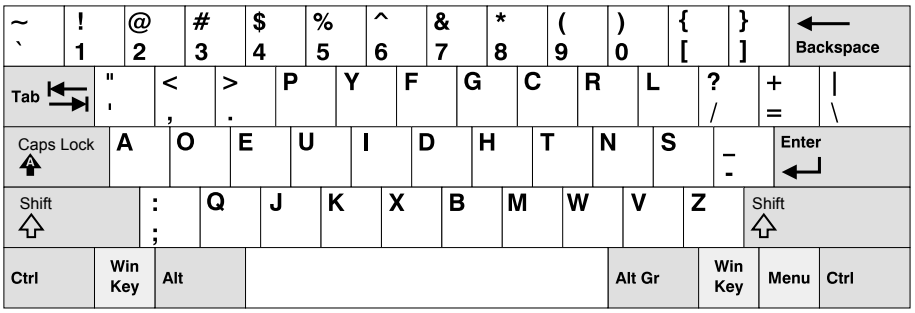
Od vynálezu tohoto psacího stroje tedy existuje rozložení klávesnice „QWERTY“, jehož název pochází od prvních šesti kláves první řady. Mnohdy bývá označeno také jako „univerzální“ rozložení. V následujících několika desetiletích vzniklo mnoho alternativních rozložení. V roce 1932 představil své rozložení profesor August Dvorak (podle toho nese název Dvorak; někdy označováno také jako „zjednodušená“ klávesnice) z Washingtonské univerzity (patentováno 1936). Tato varianta umisťuje písmena (klávesy) podle četnosti v anglickém textu. To umožňuje psát vyšší rychlostí a s menší chybovostí než s rozložením QWERTY. Přesto však je QWERTY nejpoužívanějším, zvláště díky tomu, že přišlo jako první a mnoho lidí na ně bylo zvyklých a novou klávesnici se nechtěli učit [8]. Porovnání těchto dvou klávesnic je na obr. 2.1 a 2.2.

Mimo tato rozložení vznikla např. německá klávesnice QWERTZ prohozením znaků Z a Y, protože písmeno Z je v němčině mnohem častější než Y. Česká klávesnice QWERTZ z ní vychází, protože první psací stroje se k nám začaly dovážet na konci 19. století v době

[1] majuskule = velká písmena, verzálky
[2] minuskule = malá písmena, minusky



Obrázek 2.1: Rozložení klávesnice QWERTY.



Obrázek 2.2: Rozložení klávesnice Dvorak.

Rakousko-Uherska. Později byla tato původně německá klávesnice doplněna písmeny s diakritikou na číselné řadě [12].

2.2 Hmatová metoda

Klávesnici je možné ovládat jedním až deseti prsty. V případě ovládání všemi deseti prsty dokonce dvěma základními způsoby (vědomě pomocí paměti nebo hmatovou metodou pomocí podvědomí). Zvládnutí psaní všemi deseti pro obsluhu počítače znamená hned několik výhod. Jedná se především o zrychlení ovládání klávesnice, tedy komunikace s počítačem. Důležité je též získání většího komfortu a pohody, neboť nemusíme neustále vyhledávat znaky na klávesách a následně kontrolovat na monitoru. Tímto získáváme výhodu soustředit se více na vkládaný text, čímž dochází i ke snížení počtu chyb. Důležité je najít optimální rovnováhu rychlosti, přesnosti a ergonomie.

Lidé píšící pouze deseti prsty zpaměti nemají možnost zjistit (bez kontroly zrakem), zda napsali chybu, neboť tu si na rozdíl od lidí, kteří používají hmatovou metodu a mají ji dostatečně zažitou, neuvědomí (chybí jim podvědomá zpětná vazba). Lidé, kteří mají hmatovou metodu řádně zažitou, napsání chyby téměř vždy ihned zjistí a mohou ji okamžitě odstranit. A to vše bez sledování klávesnice i obrazovky, neboť tato metoda neznamená jen psaní bez dívání se na klávesnici, ale i bez dívání se na obrazovku. To však předpokládá psaní s minimálním počtem chyb. Psát tak, že je nutné neustále opravovat (je nutné přerušit psaní a provést opravu, což vede k „přetržení“ myšlenky, neboť provedení opravy je vždy vědomé) nebo po napsání vyhledávat chyby, je velmi neefektivní.

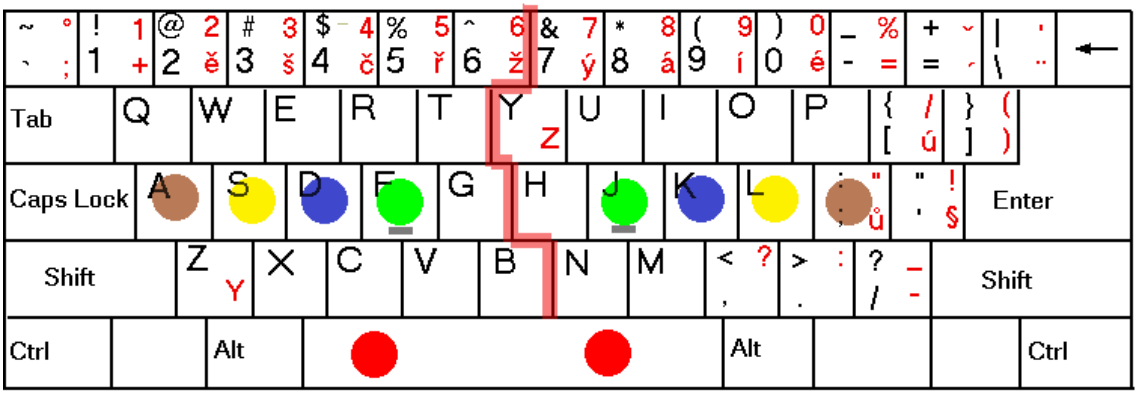
Způsob generování znaků pomocí hmatové metody umožňuje více se soustředit na obsah

svých myšlenek a přesnost jejich vyjádření, neboť osvobozuje od přemýšlení nad způsobem komunikace s počítačem. Díky zvládnutí této metody dojde k uvolnění tvůrčího potenciálu pro vlastní tvorbu (text, program atd.). Ta se díky tomu stane propracovanější a bude přesněji vyjadřovat autorovy myšlenky.

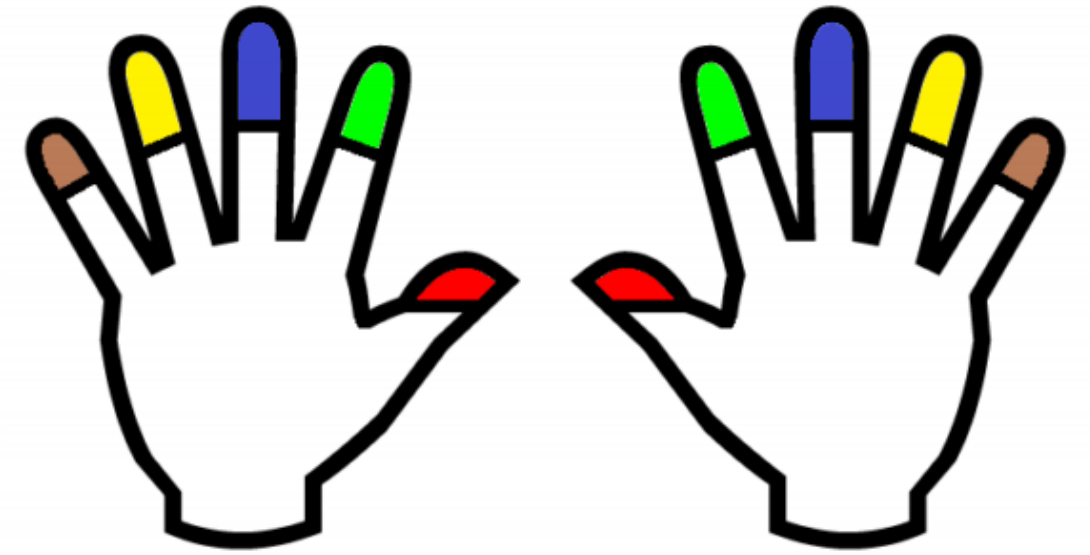
Výhoda je tak velká, že ovládání klávesnice pomocí hmatové metody se stane efektivnější i v některých případech, kdy by jinak bylo výhodnější použít myš. Lze proto říci, že hmatová metoda není výhodná jen pro psaní, ale i pro komunikaci s počítačem a navíc pomocí podvědomí, tedy jinak při ovládání klávesnice nevyužité kapacity člověka.

Významnou roli hraje tento způsob ovládání klávesnice v ochraně zdraví obsluhy. Výhodou je rozložení námahy na všech deset prstů, snížení zrakové únavy nebo odstranění bočního vytáčení páteře. V případě, kdy opisujeme předlohu a není nutné sledovat obrazovku, je možné umístit opisovaný text kolmo před uživatele a tím omezit předklon a pohyb hlavy. Při výuce psaní všemi deseti nejde jen o osvojení prstokladu. K této výuce patří i správný způsob sezení, držení rukou, základní poloha, psaní naslepo, technika stisku kláves a mnohé další.

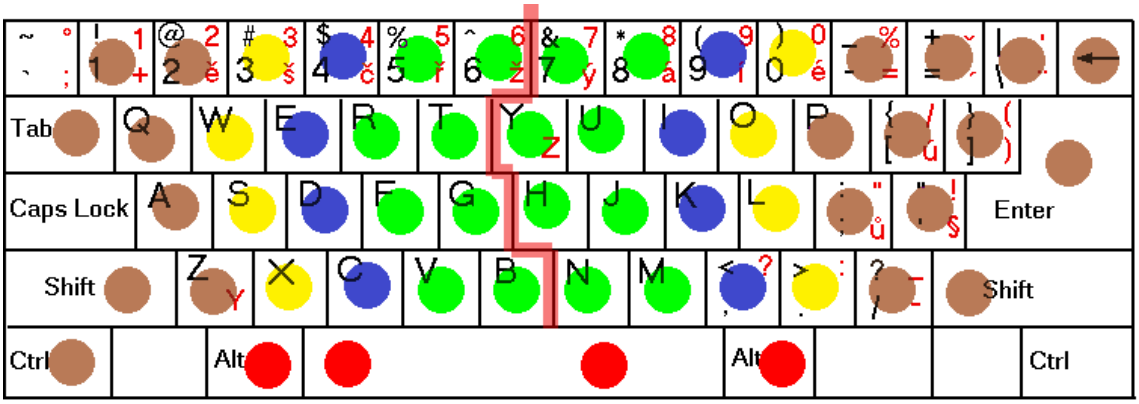
Klávesnice osobního počítače je příliš rozsáhlá na to, aby byla celá ovládána všemi deseti prsty najednou. Proto je nezbytné ji rozdělit na tři dílčí klávesnice. Psaní znaků se v každé dílčí klávesnici provádí z jiné polohy prstů. Tuto polohu nazýváme *základní poloha*. Výhodu pro nácvik psaní všemi deseti mají lidé, kteří se někdy učili na klávesový hudební nástroj. Naopak problémy s nácvikem mohou mít osoby, které se naučily ovládat klávesnici jen několika prsty a mají tento systém již „zažitý“, neboť návyk nutí člověka například dívat se na klávesnici atd. Velice nevýhodné je, ovládá-li někdo klávesnici všemi deseti prsty, ale ne hmatovou metodou. Daleko snazší je osvojit si správný nácvik od začátku, než později se odnaučit špatný způsob ovládání. Při osvojení si správných návyků se rychleji získá dovednost ovládání klávesnice a dosáhne se lepších výsledků. Ovládání všemi deseti prsty je vhodné jak pro praváky, tak i pro leváky. Většinou leváci píší lépe než praváci. Nesnáze při výuce mohou mít dyslektici. Pro psaní naslepo desetiprstovou hmatovou metodou na největší části klávesnice PC, jež je podobná psacímu stroji, tedy na alfanumerické dílčí klávesnici, je nutné mít ruce umístěny stále v jedné výchozí poloze, která zabezpečí možnost pohybu prstů po celé této dílčí klávesnici naslepo. Tuto polohu prstů na jednotlivých klávesách nazýváme základní polohou (viz obr. 2.3 a 2.4). Standardní prstoklad je zobrazen na obr. 2.5.



Obrázek 2.3: Základní poloha prstů při standardním prstokladu.



Obrázek 2.4: Základní poloha prstů při standardním prstokladu.



Obrázek 2.5: Standardní prstoklad.

Zásady správného psaní

K tomu, abychom si správně osvojili psaní všemi deseti, musíme zvládnout nejenom správný prstoklad a techniku mačkání kláves, ale i správné držení celého těla.

Při psaní všemi deseti na klávesnici počítače nebo elektrického psacího stroje jsou prsty jen mírně ohnuty, kláves se dotýkají svými bříšky a při psaní jemně kloužou z klávesy na klávesu. Prsty i zápěstí by se neměly prohýbat. Podélná osa prostředníku má být téměř kolmá na jednotlivé řady znaků na klávesnici. Dlaně se neopírají o hranu klávesnice ani stolu nebo o opěrku. O jednotlivé údery se stará pouze prst, nikoliv celé zápěstí. Ostatní prsty zůstávají vždy v základní poloze. Po napsání příslušného znaku se prst opět vrací do základní polohy.

Paže jsou volně podél těla a jsou ohnuty v loktu do mírně stoupajícího úhlu, shodného s úhlem sklonu klávesnice, tedy 10 ° až 15 ° od vodorovné polohy. Ruce jsou umístěny v základní poloze na klávesnici počítače. Zápěstí se nikdy neopírá o rám klávesnice či o hranu stolu. Opěrky před klávesnicí počítače včetně těch, jež jsou součástí klávesnice, nelze při psaní desetiprstovou metodou použít. Zápěstí musí být v rovině a celá ruka se pohybuje co nejméně. Prsty mají klouzat z klávesy na klávesu.

K dodržení pravidel pro správné umístění paží je nezbytné dodržet požadavky na správné sezení na vhodné židli, kdy záda jsou opřena o zádovou opěrku, především v bederní části. Výška sedadla má být nastavena tak, aby nohy v kolenou svíraly pravý úhel a chodidla byla celou plochou na podlaze. Nohy není vhodné křížit. Možno je jednu nohu povysunout dopředu, ale při dlouhodobém sezení je nejlépe mít nohy od sebe (úhel až 45 °). Při psaní vsedě má být dodržena dostatečná vzdálenost očí od obrazovky – cca 50 až 80 cm.

Nedodržení těchto zásad může vést k mnohým zdravotním obtížím, například k bolestem zad, bolestem na hrudní kosti podobným infarktu myokardu atd. [3].

2.3 Faktory ovlivňující psaní

Jedním ze základních faktorů ovlivňujících rychlost a přesnost psaní všemi deseti je rozložení prstů na klávesnici. Na obrázcích 2.4 a 2.5 je zobrazeno, které prsty ovládají konkrétní klávesy. Vidíme, že ukazováček obsluhuje 8 kláves (skupina), ale v relativně malém prostorovém rozmezí. Dá se tedy předpokládat, že stisk těchto kláves bude poměrně rychlý. Komplikace a zpomalení může nastat obzvláště v případech, kdy je nutné stisknout ukazováčkem více kláves ze stejné skupiny po sobě (tzn. bez použití jiného prstu mezitím, např. levým ukazováčkem text „*num*“). Na druhou stranu malíček obsluhuje také velké množství kláves. Ty se ale nachází poměrně daleko a je nutné prst „natahovat“. Navíc se jedná o znaky používané zřídka oproti ostatním, a proto bude stisk těchto kláves pravděpodobně nejpomalejší. Četnost používání znaku a navyknutní na jeho stisknutí bude zajisté také jedním z faktorů ovlivňující rychlost jeho stlačení. Další dva prsty (prostředníček a prsteníček) obsluhují oba po čtyřech znacích. Očekáváme tedy velikou rychlost stisku a malou chybovost. Posledním prstem, který zajišťuje psaní všemi deseti, je palec. Ten ale obstarává pouze klávesy *Alt* a *mezerník*, tedy klávesy z hlediska monitorování nezajímavé.

Další věc, která může mít souvislost s výkonností psaní, je denní doba. Někteří lidé budou podávat nejlepší výsledky po ránu, kdy jsou „čerství“ a mají dostatek energie. Jiní budou potřebovat nějakou dobu na rozepsání a budou mít nejlepší výkon v odpoleních hodinách. Ostatním může vyhovovat večerní klid. Proto je nutné počítat i s touto okolností a zkoumat, jakou má na psaní vliv.

Kapitola 3

Keylogging a možnosti tvorby GUI ve Windows

V této kapitole se věnuji *keyloggingu*, tedy akci, při které jsou zaznamenávány stisknuté klávesy. Blíže se zaměřím na popis toho, jak funguje komunikace aplikací v systému Windows. Dále popíšu, jaké jsou možnosti zachycování kláves a jakým způsobem fungují. Další částí této kapitoly je rozbor možností tvorby grafického uživatelského rozhraní a grafů ve Windows. Tato kapitola čerpá především z [4], [1] a [2].

3.1 Zprávy ve Windows

Na rozdíl od aplikací založených na systému MS-DOS, aplikace založené na Windows jsou řízené událostmi. K získání dat nevolají explicitně funkce, místo toho čekají, dokud jim systém data nepředá.

Systém předává všechna data pro aplikaci různým oknům v aplikaci. Každé okno má funkci nazývanou *procedura okna* (*window procedure*) (dále jen PO), kterou systém volá, kdykoli má data pro okno. PO zpracuje data a řízení vrátí systému.

Předávání dat proceduře okna je ve formě zprávy (*message*). Zprávy jsou generované jak systémem, tak aplikací. Systém generuje zprávy pro každou vstupní událost, např. psaní na klávesnici, pohyb myši, kliknutí na nějaký prvek atd. Aplikace může generovat zprávy přímo pro svá vlastní okna, aby zajistila vykonání potřebných úloh, nebo může komunikovat s okny jiných aplikací.

Systém zasílá zprávu proceduře okna se čtyřmi parametry: handle okna, identifikátor zprávy a dvě hodnoty nazývané parametry zprávy. Handle označuje okno, pro které je zpráva určena a systém je využívá pro zjištění, která PO má tuto zprávu přijmout.

Identifikátor zprávy značí její účel. Pokud PO zprávu přijme, použije její identifikátor k určení způsobu, jak zprávu zpracovat. Například identifikátor zprávy `WM_PAINT` znamená, že se změnila klientská oblast okna a musí být překreslena.

Parametry zprávy představují data nebo ukazatel na data, které jsou použity procedurou při zpracování zprávy. Význam a hodnota těchto parametrů se liší u každé zprávy a mohou být jakýchkoli typů (celočíselný typ, ukazatel na strukturu, ...). Pokud je zpráva nevyužívá, jsou obvykle nastaveny na hodnotu `NULL`. PO tedy musí zkontrolovat identifikátor zprávy a podle toho zjistí, jak má parametry zpracovat.

3.1.1 Typy zpráv

Existují 2 typy zpráv:

- Systémem definované zprávy (System-Defined Messages)
- Aplikací definované zprávy (Application-Defined Messages)

Systémem definované zprávy

Systém zasílá systémem definované (dále jen SD) zprávy, pokud komunikuje s aplikací. Používá tyto zprávy k ovládání operací aplikací a k zprostředkování dat a jiných informací ke zpracování. Aplikace tyto SD zprávy mohou zasílat také. Využívají je především k ovládání operací řídících oken.

Každá SD zpráva má jedinečný identifikátor a odpovídající symbolickou konstantu (definovanou v hlavičkovém souboru SDK), která odpovídá účelu zprávy (`WM_PAINT`). Prefix konstanty určuje kategorii, do které zpráva patří. Nejčastější zprávy jsou z kategorie s prefixem `WM`, které značí obecnou kategorii. Ta pokrývá širokou škálu zpráv jako stisk klávesy, pohyb myši, vytvoření okna atd. Mezi další kategorie patří např. `CB` (ComboBox), `LB` (ListBox), `TB` (Toolbar). Celý výčet je možné najít v [1].

Aplikací definované zprávy

Aplikace může vytvářet zprávy, které jsou využity vlastními okny nebo slouží ke komunikaci s okny jiných procesů – aplikací definované zprávy. Pokud aplikace vytváří vlastní zprávy, PO, která je přijímá, je musí interpretovat a náležitě zpracovat.

3.1.2 Směrování zpráv

Systém používá 2 metody ke směrování zpráv do procedury okna. Zasíláním zpráv do FIFO fronty vzniká *fronta zpráv* (*message queue*) - objekt v paměti, který dočasně ukládá zprávy a zasílá je přímo do procedury.

Zpráva, která je uložena do fronty zpráv, je označována jako *frontová zpráva* (*queued message*). Jsou to převážně vstupní zprávy od uživatele zaslané skrze myš nebo klávesnici, jako `WM_MOUSEMOVE`, `WM_LBUTTONDOWN`, `WM_KEYDOWN` a `WM_CHAR`. Další souvisí s časovačem, kreslením a ukončením - `WM_TIMER`, `WM_PAINT` a `WM_QUIT`. Většina ostatních zpráv, které jsou zaslány proceduře přímo, jsou označovány jako *nefrontové zprávy* (*nonqueued messages*).

3.1.3 Zpracování zpráv

Aplikace musí odstranit a zpracovat zprávy zaslané do fronty zpráv jejího vlákna. Jednovláknová aplikace většinou používá *smyčku zpráv* (*message loop*) v její hlavní funkci. V té se odstraňují a zasílají zprávy ke zpracování odpovídající proceduře okna. Aplikace s více vlákny mohou zahrnout smyčku zpráv do každého vlákna, které vytváří okno.

Smyčka zpráv

Jednoduchá smyčka zpráv volá právě jednou tyto funkce: `GetMessage`, `TranslateMessage` a `DispatchMessage`. Pokud nastane chyba, `GetMessage` vrací hodnotu -1:

```
MSG msg;
BOOL bRet;
while ((bRet = GetMesage(&msg, NULL, 0, 0)) != 0)
{
     if (bRet == -1)
     {
          // handle the error and possibly exit
     }
     else
     {
          TranslateMessage(&msg);
          DispatchMessage(&msg);
     }
}
```

Funkce `GetMessage` se stará o vyjímání zpráv z fronty. Vrací nenulovou hodnotu, pokud se nejedná o zprávu `WM_QUIT` – v takovém případě je smyčka ukončena. V jedno-vláknové aplikaci je ukončení smyčky často prvním krokem k zavření aplikace. Pokud je specifikován handle okna jako druhý parametr funkce, z fronty jsou odebírány zprávy příslušné tomuto oknu.

Smyčka zpráv ve vlákně musí obsahovat funkci `TranslateMessage`, pokud toto vlákno přijímá vstup z klávesnice. Systém generuje *virtuální klávesové* (*virtual-key*) (dále jen VK) zprávy (`WM_KEYDOWN` a `WM_KEYUP`) pokaždé, když uživatel stiskne klávesu. VK zpráva obsahuje VK kód, který identifikuje, která klávesa byla stisknuta, ne však hodnotu znaku. K získání této hodnoty je potřeba právě funkce `TranslateMessage`, která přeloží VK zprávu na *zprávu znaku* (`WM_CHAR`) a vloží ji zpět do fronty.

Funkce `DispatchMessage` zasílá zprávu do PO.

Hlavní vlákno aplikace spouští smyčku zpráv po inicializaci aplikace a vytvoření alespoň jednoho okna. Po startu smyčka přijímá zprávy z fronty příslušného vlákna a zasílá je příslušnému oknu. Smyčka končí v okamžiku, kdy funkce `GetMessage` vyjme z fronty zprávu `WM_QUIT`.

Pro jednu frontu zpráv existuje vždy jedna smyčka. To i v případě, že aplikace obsahuje několik oken. Funkce `DispatchMessage` vždy zašle zprávu vhodnému oknu.

3.2 Hákování klávesnice

Hákování (*hooking*) je mechanismus, díky kterému je aplikaci umožněno zachycovat události jako zprávy, akce myši (pohyb, kliknutí) a stisky kláves. Funkce, která zachycuje tyto typy událostí, se nazývá *hákovací procedura* (*hook procedure*) (dále jen HP) (viz kap. 3.2.2). HP může reagovat na každou událost, kterou přijme, a může ji pozměnit nebo zahodit. Hákování má tendenci zpomalovat systém, protože se zvyšuje počet zpracování každé zprávy.

3.2.1 Hákovací řetězy

Systém podporuje mnoho typů *háků* (*hooks*). Aplikace může např. použít hák `WH_MOUSE` ke sledování zpráv generovaných myší.

Systém obsluhuje pro každý typ háku vlastní *hákovací řetěz* (*hook chain*). Hákovací řetěz je seznam ukazatelů na speciální, aplikací definovanou, funkci - HP. Pokud se objeví

zpráva, která je asociována s konkrétním typem háku, systém předá zprávu do každé HP odkazované v hákovacím řetězu. Pro některé typy háků může HP pouze monitorovat zprávy, jinak může zprávy měnit nebo zastavit jejich postup skrz řetěz, aby zabránila dosažení další HP nebo cílového okna.

3.2.2 Hákovací procedury

Aby bylo možné využít konkrétních typů háků, je nutné vytvořit HP a použít funkci `SetWindowsHookEx`, která zajistí připojení k řetězu (na jeho začátek) asociovanému s tímto hákem. Když nastane hákovaná událost, systém zavolá proceduru na začátku řetězu. Procedura předá událost následující proceduře zavoláním funkce `CallNextHookEx`.

Níže je zobrazena ukázka hákovací procedury `HookProc`. Parametr `nCode` je kód háku, který procedura používá k rozhodnutí, jakou akci má vykonat. Jeho hodnota záleží na typu háku – každý typ má vlastní množinu kódů. Hodnoty parametrů `wParam` a `lParam` obvykle obsahují informace o zprávě, která byla zaslána:

```
LRESULT CALLBACK HookProc(int nCode, WPARAM wParam, LPARAM lParam)
{
     // process event
     ...

     return CallNextHookEx(NULL, nCode, wParam, lParam);
}
```

Globální hák (*global hook*) sleduje zprávy pro všechna vlákna. *Vláknově-specifický hák* (*thread-specific hook*) monitoruje zprávy pouze vlastního vlákna. Globální háková procedura může být zavolána z kontextu jakékoli aplikace, proto musí být v odděleném DLL modulu. Vláknově-specifická procedura je volána jen v kontextu vlastního vlákna a může být tedy ve stejném modulu jako aplikace (ale nemusí).

3.2.3 Typy háků

Každý typ háku dovoluje aplikaci sledovat jiné zprávy. Pro účely této práce je využit hák `WH_KEYBOARD_LL`, který umožňuje sledovat vstupní události klávesnice. Mimo tento existují např. háky `WH_GETMESSAGE`, `WH_KEYBOARD`, `WH_MOUSE_LL` a `WH_MOUSE`.

3.3 Možnosti tvorby GUI ve Windows

V této části se zaměřím na popis nejznámějších prostředí, ve kterých lze vytvořit grafické uživatelské rozhraní ve Windows. Jejich přehled ukazuje tabulka 3.1.

Win32 API

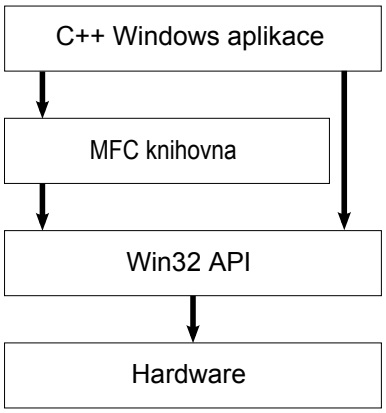
Jednou z možností, jak vytvořit GUI ve Windows, je použít Windows API (Application Programming Interface). Win API představuje nejnižší úroveň komunikace mezi aplikací a operačním systémem. S tím souvisí obtížnost programování, příliš mnoho kódu, ne-objektově orientované programování, atd. Proto není pro náš účel, kde potřebujeme tvořit složité kontrukce, příliš vhodné. Ostatní prostředí jsou však nějakým způsobem nadstavbou nad Windows API.

Technologie	Programovací jazyk
Win32 API	C
MFC	C++
QT	C++
AWT, Swing	Java
WinForms	jazyky .NET
WPF	jazyky .NET

Tabulka 3.1: Přehled technologií pro tvorbu grafického uživatelského rozhraní ve Windows.

MFC

Microsoft Foundation Class Library neboli MFC je knihovna obsahující C++ třídy, které zaobalují různé funkce a části Windows API. Oproti Windows API jsou tedy výhody zřejmé – rychlejší a pohodlnější vývoj díky objektově orientovanému programování. Vztah mezi těmito dvěma prostředími je zobrazen na obr. 3.1. I přes tyto výhody je tato technologie už poměrně zastaralá, a proto jsem se pro ni nerozhodl.



Obrázek 3.1: Vztah mezi Windows API a MFC.

QT

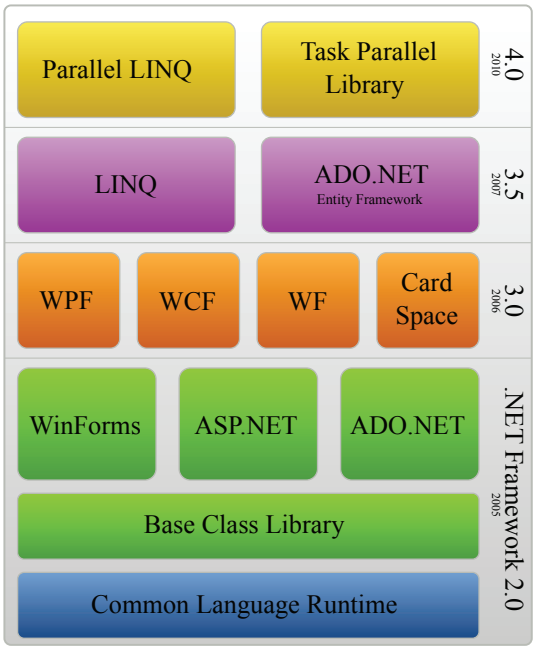
Další možností tvorby GUI v jazyku C++ je použití QT Frameworku. Existuje však i portování na ostatní jazyky jako např. Python, Ruby, Java nebo C#. Tento způsob je však komplikovaný.

AWT, Swing

Abstract Windows Toolkit neboli AWT a Swing jsou dvě prostředí pro tvorbu grafického uživatelského rozhraní v Jave (Swing má podobný princip jako AWT, ale je propracovanější, osahuje více prvků a umožňuje přesnější nastavení vzhledu aplikace). Oproti MFC nebo QT jde tedy zase o větší úroveň abstrakce a zjednodušení programování. V našem systému je ale potřeba volat několik API funkcí systému Windows (viz kap. 5.2), což není v Jave moc pohodlné. Proto jsem se zaměřil na jazyk C#.

WinForms

Windows Forms neboli WinForms je grafické API podobné AWT nebo MFC. Je součástí .NET Frameworku od Microsoftu, jehož struktura je na obr. 3.2. Díky vrstvě Common Language Runtime (CLR) je možné použít pro implementaci jakýkoli .NET jazyk (C#, VB.NET, J#, ...) – zdrojový kód je totiž zkompilován do byte kódu a za běhu aplikace vykonáván pomocí CLR. Další výhodou je pohodlný UI designer zabudovaný ve Visual Studiu. Díky těmto vlastnostem a podobnosti s MFC, se kterým jsem měl zkušenosti, jsem se rozhodl využít toto rozhraní pro tvorbu naší aplikace.



Obrázek 3.2: Struktura .NET Frameworku.

WPF

Nejpokročilejší technologií pro tvorbu GUI je Windows Presentation Foundation (WPF). Je považován za nástupce WinForms. Stejně tak je součástí .NET Frameworku, z čehož plynou stejné výhody jako v případě WinForms. Pro definici uživatelského rozhraní využívá jazyk XAML – deklarativní značkovací jazyk (jedná se vlastně o jazyk XML) s danou syntaxí a sémantikou. Díky tomuto přístupu je možné oddělit logiku (funkčnost) aplikace od jejího vzhledu.

3.3.1 Knihovny pro tvorbu grafů

V našem systému se většina vyhodnocených informací zobrazuje pomocí grafů. Pro tvorbu grafů existuje obrovské množství knihoven (zvláště pro Javascript). Já jsem se zaměřil pouze na ty, které je možné využít v desktopových aplikací. Bohužel mnoho z nich nevypadá příliš dobře. Povedené a hezky vypadající knihovny jsou na druhou stranu placené (jejich přehled můžeme vidět v tabulce 3.2). Z toho důvodu jsem se rozhodl využít vestavěné třídy .NET Frameworku ze jmenného prostoru `System.Windows.Forms.DataVisualization.Charting`,

a to především třídu `Chart` představující komponentu grafu. Kromě použití nativního jazyka pro implementaci těchto grafů je výhodou i to, že knihovnu není nutné nijak integrovat do projektu.

Knihovna	**Programovací jazyk**
Codejock Chart Pro	C++
ChartDirector	jazyky .NET, Java, PHP, Perl, Python, Ruby, C++
IOCOMP	jazyky .NET
.NET Charting	jazyky .NET
RadChart	jazyky .NET

Tabulka 3.2: Přehled knihoven pro tvorbu grafů.

Kapitola 4

Návrh vyhodnocování informací o psaní na klávesnici

Jak bylo řečeno v úvodu této práce (1), naše aplikace zaznamenává všechny stisknuté klávesy ve Windows a z těchto informací vyhodnocuje různé údaje, které mohou být uživateli nápomocny ve zlepšení svých psacích schopností. V této kapitole se budu věnovat popisu toho, jakým způsobem jsou stisknuté klávesy zpracovávány a jaké výsledky je možné z těchto údajů získat.

4.1 Segmentace napsaného textu

Při psaní uživatele na klávesnici jsou zaznamenávány všechny stisknuté znaky a napsaný text. Klíčovým způsobem jak dělit text na ucelené části je rozpoznávání sekcí – *segmentace*. Veškeré zaznamenávané údaje ze stisků kláves se vztahují vždy k jedné aktuální sekci, a to do té doby, dokud nedojde k její změně. Pokud tak nastane, je sekce vložena do databáze, aby ji bylo možné v případě potřeby vyhodnotit (viz kap. 5). Ke změně sekce může dojít dvěma způsoby:

- přepnutí aktivního okna,
- nečinnost uživatele.

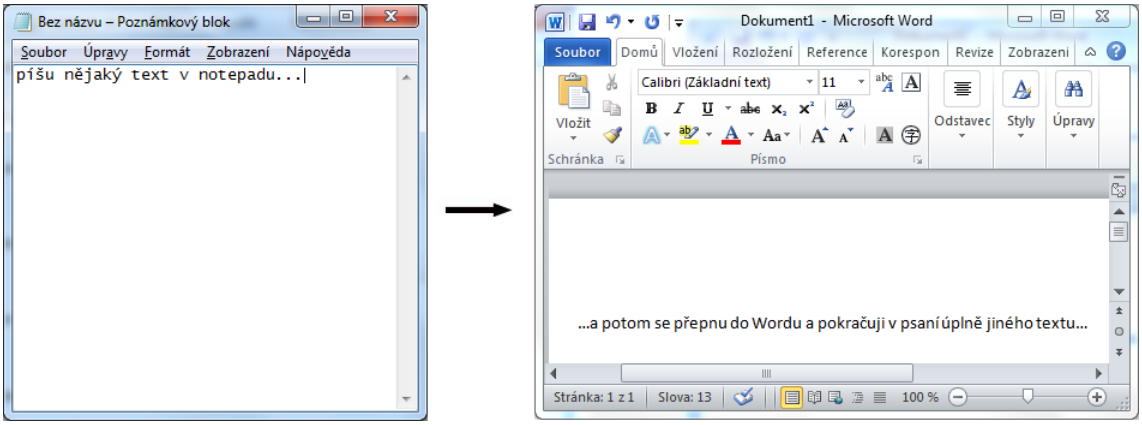
Přepnutí aktivního okna

Prioritním ukazatelem změny sekce v naší aplikaci je přepnutí aktivního okna[1] (viz obr. 4.1). Při každém stisku klávesy se zjišťuje název aktivního okna a kontroluje se, zda je předchozí název různý od aktuálního. Tzn., že pokud uživatel píše např. článek na web a přepne se na ICQ a začne psát, v tu chvíli dochází ke změně sekce. Ke změně tedy dochází až v okamžiku, kdy uživatel začne v nové aplikaci něco psát. Pokud by si uživatel pouze zobrazil zprávu na ICQ a pak pokračoval v psaní článku, ke změně sekce nedojde.

V těchto případech se jedná zároveň o změnu aplikace. Mohou nastat i situace, kdy dojde ke změně sekce, přitom ke změně aplikace ne. Jedná se především o různé vyskakovací dialogy. Do této kategorie spadají např. i dialogy pro zadávání hesel. Nejen z tohoto důvodu je proto výhodné mít založenou segmentaci na změně aktivního okna na rozdíl od změny

[1] aktivní okno = okno na popředí, má focus

aplikace. Tato informace (název aplikace, ke které okno patří) je ale naším systémem také získávána a je později použita pro hlavní slučování sekcí (viz kap. 5).



Obrázek 4.1: Změna sekce prostřednictvím přepnutí aktivního okna (v tomto případě se jedná i o změnu aplikace). Ke změně dochází v okamžiku stisku první tečky ve Wordu.

Nečinnost uživatele

K ukončení sekce (ne přímo změně) dojde také v případě, že uživatel nějakou dobu žádnou klávesu nestiskne. Nastane stav nečinnosti, který je přerušen v momentě stisku nějaké klávesy. Dojde tím k vytvoření sekce nové.

Tato vlastnost přispívá k lepší škálovatelnosti výsledků vyhodnocení. Pokud např. uživatel píše nějaký článek (a během psaní nedochází ke změnám sekce kvůli přepnutí okna) a v průběhu si odskočí na oběd a poté bude v práci pokračovat, bude tento text rozdělen do dvou (může být i více, pokud se v psaní objevují kratší přestávky) sekcí. Díky tomu může být porovnána např. efektivita před obědem a po obědě.

Každou sekci charakterizuje několik vlastností:

- název aplikace,
- název titulku okna,
- čas psaní,
- obsažený text,
- počet slov,
- překlepové chyby (seznam chyb),
- rytmičnost psaní (seznam písmen a jejich rychlostí stisku) a
- počet úhozů za minutu.

Aby bylo možné pro každou sekci určit časový rozsah psaní, je při každém stisku klávesy zaznamenán aktuální čas. Není vhodné zjistit aktuální čas až po skončení sekce, neboť ta může být ukončena i po nějaké době, co uživatel stiskl poslední klávesu (pokud by došlo k přepnutí sekce kvůli nečinnosti uživatele). Koncovým časem se tedy stává poslední zaznamenaný čas (čas stisku poslední klávesy).

4.2 Faktory ovlivnující vyhodnocení

Je téměř jisté, že ne všechny údery kláves nám prozradí psací schopnosti uživatele (jak bylo psáno v kapitole 4.1). Proto jsou některé sekce ignorovány a vyhodnocen je pouze zbytek. Faktory k rozhodnutí jsou:

- počet slov v sekci,
- počet písmen v sekci,
- zda je aplikace odfiltrovaná.

Vyhodnocené tedy budou jen ty úseky, které obsahují požadovaný minimální počet slov a písmen. Pokud bychom brali v úvahu pouze počet slov, mohly by k vyhodnocení „proklouznout“ i sekce, které představují nárazové psaní a nikoli souvislý text, jako např. opravy chyb v celém dokumentu.

Filtrování aplikací představuje způsob, jak získávat žádoucí a relevantní výsledky vyhodnocení. Uživatel při běžné práci s počítačem používá několik aplikací. Stisknuté klávesy jsou zaznamenávány pro každou z nich. Přitom výsledky psaní, které ho budou zajímat, se budou skládat pouze z určitých aplikací – těch, které mají o kvalitách psaní nějakou vypovídající hodnotu. V praxi nás například budou zajímat výstupy z Wordu, napsaný článek na blog použitím nějakého prohlížeče či psaní e-mailů skrz některého e-mailového klienta. Naopak vyhledávání na internetu, hraní her, programování či několikaslovná odpověď na chatu příliš nereflektují uživatelovy psací schopnosti, a tudíž pro nás budou nezajímavé. Uživatel má volbu zrušit zařazení takovýchto aplikací do vyhodnocení. Nasnímaná data z těchto aplikací budou sice do databáze uložena, ale ve výsledku se nezobrazí. Pokud se uživatel po nějaké době rozhodne, že chce vidět i tyto výsledky, data budou k dispozici. Druhá možnost je vypnutí logování takovéto aplikace. V tomto případě data budou zahozena a do databáze se vůbec neuloží. Samozřejmě data uložená před vypnutím logování také ve vyhodnocení zobrazena nebudou.

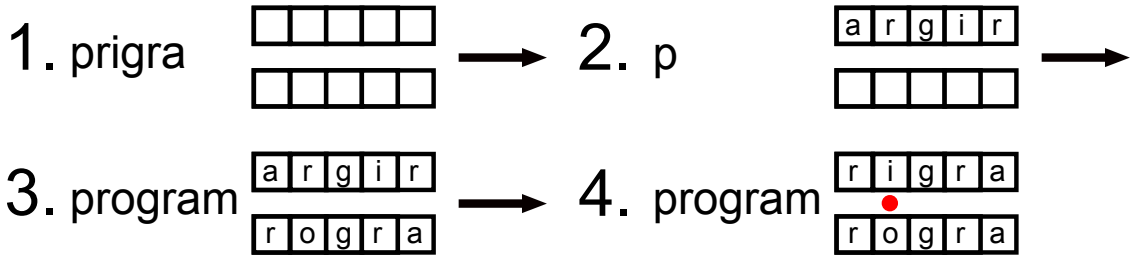
Možnou bezpečnostní dírou této aplikace je fakt, že mohou být zaznamenána i citlivá data nebo různá hesla. Veškerý namonitorovaný text je ale uložen pouze v databázi. Uživateli se zobrazují už vyhodnocené výsledky a k samotnému textu přístup nemá. Hesla navíc bývají většinou jednoslovná, takže aplikací budou zahozeny z důvodu nesplnění minimálního počtu slov potřebných k vyhodnocení.

4.3 Detekce chyb

Jedním z přínosů vytvořené aplikace by mělo být upozornění na časté překlepové chyby ve psaní.

Chyby vzniklé překlepem při rychlém psaní jsou nejčastěji opravovány tak, že uživatel odmazává doposud napsaná písmena pomocí klávesy BACKSPACE, dokud nesmaže i chybný znak nebo znaky. Při každém stisku této klávesy jsou odstraněné znaky zaznamenávány do seznamu chyb. V momentě, kdy uživatel stiskne jinou klávesu než BACKSPACE, jsou nové znaky ukládány do seznamu opravných znaků. V případě, že je dokončeno celé slovo (můžeme předpokládat, že chyba byla opravena), dojde k hledání chyby. Nejprve se reverzuje seznam s chybnými znaky, protože mazané znaky jsou ukládány v opačném pořadí než opravné (mažeme slovo od konce). Poté se celý seznam prochází a hledá se první neshoda s odpovídajícím znakem v seznamu opravných znaků. Vždy je tedy za chybný znak

považován ten, který měl na dané pozici být. Typicky jde o první znak, protože smažeme pouze tolik znaků, abychom odstranili chybu. Proto bychom automaticky mohli brát jako chybu první znak v seznamu. Může se ale stát, že smažeme jeden či více znaků navíc, tedy i ty správné. Proto rozdílnost znaků kontrolujeme a při prvním nálezu je zazanamenán chybný znak a slovo, ve kterém k chybě došlo. Celý postup je zobrazen na obrázku 4.2.



Obrázek 4.2: Ukázka vyhodnocení chyby na slově *program*. V každém kroku je zobrazen napsaný text, nahoře seznam s chybnými znaky, dole seznam s opravnými znaky. V kroku 1 uživatel udělal ve slově chybu a všiml si jí, až když měl napsáno *prigra*. Začal tedy odmazávat až po písmeno *p* a mazané znaky se ukládaly do seznamu chyb (krok 2). Ve 3. kroku se opravil a dopsal slovo *program* – nové znaky jsou uloženy do seznamu opravných znaků. Je dokončeno slovo, takže je otočen seznam s chybnými znaky a kontroluje se první neshoda (krok 4). Ta je nalezena u písmena *o*, takže je zaznamenáno jako chybný znak.

Bohužel tento systém nezachytí chyby opravované jiným způsobem, jako např. posunutí kurzoru pomocí šipek a následné mazání textu, nahodilé opravy v dokumentu, označení textu myší a následné úpravy atd. Tyto způsoby jsou ale u hmatové metody ne často používané, tudíž nepředstavují veliký problém.

Při grafickém zobrazování nejchybnějších znaků jsou brány v úvahu pouze písmena a ostatní znaky (jako číslice, interpunkce) jsou odstraněny, protože nás příliš nezajímají. Rovněž jsou všechna velká písmena převedena na malá, neboť se vlastně jedná o tutéž klávesu na klávesnici.

4.4 Rytmičnost psaní

V každé sekci je zaznamenáván příchod asynchronních událostí – stisky a uvolnění kláves. Po uvolnění každé klávesy jsou spuštěny stopky a běží tak dlouho, dokud nedojde ke stisku jiné klávesy. Změřený čas je zaznamenán a hodnota určuje časový rozestup mezi těmito klávesami. Jedná se tedy o čas potřebný ke stisku odpovídající klávesy. Po uvolnění této klávesy jsou stopky vyresetovány a měří se čas zase pro následující klávesu atd. Pro každý napsaný znak tedy známe dvě složky:

- o jaké písmeno se jedná a
- jak dlouho trvalo, než byla stisknuta klávesa odpovídající konkrétnímu znaku.

Vynesením těchto písmen a jejich rychlostních údajů do časové osy získáme rytmičnost psaní na klávesnici.

Z těchto údajů ale můžeme získat mnohem více informací. Pokud bychom odstranili pomyslnou vazbu mezi sekcí a rytmičností v sekci a vzali v úvahu všechny údaje o rychlostech

stisků jednotlivých kláves ze všech sekcí, můžeme z nich získat např. údaje o nejpomalejších či nejrychlejších znacích. Stačí sečíst všechny hodnoty časů stejných písmen a poté vydělit počtem těchto stejných písmen. Získáme tím aritmetickou hodnotu rychlosti stisku znaku. Nejpomalejší znaky získáme seřazením výsledků všech písmen od největší hodnoty po nejmenší, pro nejrychlejší znaky uděláme seřazení opačné.

Namapováním znaků na konkrétní prsty, které se starají o stisky těchto znaků (viz obr. 2.4 a 2.5), můžeme získat časové údaje o stisku kláves pro každý prst zvlášť (stejným způsobem jako údaje o nejpomalejších či nejrychlejších znacích).

Některé znaky s diakritikou, které nejsou na klávesnici obsaženy přímo (např. ň, ď, ť), nejsou do tohoto vyhodnocení započítány. K jejich napsání je totiž potřeba stisknout klávesy dvě, a proto by pravděpodobně byla rychlost těchto znaků nejpomalejší a porovnání by neodpovídalo.

4.5 Počet úhozů za minutu

Úloha zjištění počtu úhozů za minutu je řešena jednoduchou rovnicí (4.1), kde k je výsledný počet úhozů, l je počet hrubých úhozů, m je počet chyb v textu, f je penalizační kvocient a t je čas představující dobu psaní (v sekundách) ([11]). Celý výraz je vynásoben 60, aby byla výsledná hodnota v jednotkách za minutu.

$$k = \frac{l - (m \cdot f)}{t} \cdot 60 \tag{4.1}$$

Počet hrubých úhozů představuje počet bodů za opsaný text. Každý znak má daný počet bodů (viz tabulka 4.1). Penalizační kvocient může být různý. Pro běžné psaní se používá hodnota 10, pro soutěže 50. V našem případě tedy použijeme hodnotu 10.

Počet bodů	**Seznam písmen**
1	aábcčdeěéfghiíjklmnopqrřsštuůúvwxyýzž,.-§)=+;
2	1234567890óABCDEFGHIJKLMNOPQRSTUVWXYZ ?:_”!/(’%<>/;¤[]\‘@#$^&*()
3	ďťňÁÉÍÓÚ{}|×÷˜
4	ĎŤŇČĚŘŠŮŽ

Tabulka 4.1: Ohodnocení jednotlivých znaků české abecedy pro výpočet hrubého počtu úhozů. Zdroj [11].

Přesnost psaní je počítána podle rovnice 4.2, kde p značí výslednou přesnost v procentech, m je počet chyb a l je počet hrubých úhozů. Výraz v závorce představuje chybovost.

$$p = 100 - (\frac{m}{l} \cdot 100) \tag{4.2}$$

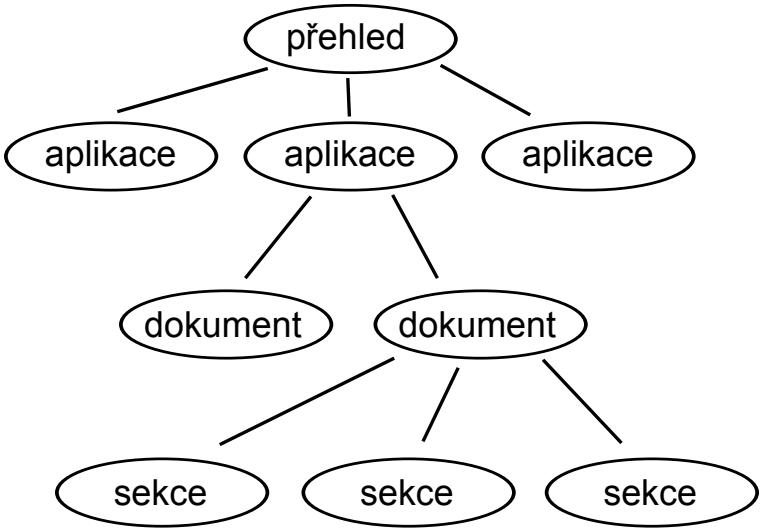
Pro získání počtu úhozů a přesnosti psaní z více sekcí je potřeba spočítat vážený průměr závisející na počtu písmen v jednotlivých sekcích. Nelze tuto hodnotu počítat aritmetickým průměrem, protože by tím byly odstraněny rozdílné váhy sekcí (jedna sekce může obsahovat 1000 písmen, druhá např. pouze 200).

4.6 Slučování sekcí

Všechny nasnímané sekce a jejich vlastnosti (viz kap. 4.1) jsou uloženy v databázi. Pokud bychom tyto informace vypsali např. v tabulce, viděli bychom pouze dlouhý seznam údajů, ze kterého by šlo velmi obtížně poznat, jakými psacími schopnostmi to vlastně vládneme. Proto je nutné z těchto informací vytvořit jakýsi přehled.

V naší aplikaci je tento přehled na vrcholu čtyřúrovňové hierarchie, přičemž každá vyšší úroveň je nadřazená nižší úrovni (obr. 4.3):

1. přehled,
2. aplikace,
3. dokument,
4. sekce.



Obrázek 4.3: Čtyřúrovňová hierarchie tvořící přehled vyhodnocení psacích schopností.

Sekci jsme si představili v kapitole 4.1. *Dokument* má ty samé vlastnosti jako sekce, ale je tvořen sloučením několika sekcí dohromady (minimálně jedna) – těch, které mají stejný název aplikace a titulku okna. Musí se tedy sloučit všechny vlastnosti. Spojení dvou sekcí pro jednotlivé vlastnosti vypadá následovně:

- Obsažený text – text druhé sekce se připojí za text první sekce.
- Název aplikace – zůstává stejný.
- Název titulku okna – zůstává stejný.
- Čas psaní – jako počáteční čas je použit počáteční čas první sekce, jako koncový čas je použit koncový čas sekce druhé.
- Počet slov – součet obou počtů.

- Překlepové chyby – připojení druhého seznamu chyb za první.
- Rytmičnost psaní – připojení druhého seznamu písmen za první.
- Počet úhozů za minutu – spočítán vážený průměr podle počtu písmen v jednotlivých sekcích.

Aplikace je tvořena několika dokumenty – těmi se stejnými názvy aplikace. Stejně tak může obsahovat 1-n dokumentů. Je opět nutné spojit všechny vlastnosti (probíhá stejným způsobem jako u dokumentu).

Přehled je tvořen několika aplikacemi. Vlastnosti jsou opět spojené stejným způsobem jako u dokumentu. Přehled navíc obsahuje seznam počtu úhozů za minutu seřazený podle času. Vynesením těchto údajů do grafu můžeme pozorovat rychlost psaní v závislosti na denním čase a zjistit tak, zda má denní doba na psaní nějaký vliv (viz kap. 2.3).

Kapitola 5

Implementace

V této kapitole se zaměřím na popis implementace aplikace a některých důležitých tříd. V závěru také udělám rozbor vyhodnocení nasnímaných dat.

5.1 Koncept aplikace

Programová část diplomové práce je vytvořena na platformě Windows ve vývojovém prostředí Visual Studio 2010. Jako implementační jazyk jsem zvolil C#4.0 [10]. Grafické uživatelské rozhraní je tvořeno ve Windows Forms [5], [9]. Pro ukládání dat je využita databáze pro SQL Server.

Program tvoří 2 moduly:

- `HookService` – stará se o zachycování kláves,
- `KeyboardSkillsAnalysis` – představuje okenní aplikaci, která zobrazuje vyhodnocené výsledky.

Výsledný program je po spuštění minimalizovaný v tray liště a monitoruje stisky kláves. Po dvojkliku se aplikace maximalizuje a zobrazuje naměřené výsledky.

5.2 Modul HookService

Základním kamenem aplikace je zachycování kláves (viz kapitola 3). Tato část je implementovaná v samostatném modulu `HookService`. Důvody oddělení do zvláštního modulu jsou popsány v kapitole 3.2.2.

V tomto modulu je pouze jedna třída `GlobalKeyboardHook`. Ta obstarává veškerou práci hákování klávesnice. Využívá několik API funkcí knihovny `user32.dll`. Abychom mohli používat tyto funkce v jazyce C#, je nutné je naimportovat pomocí PInvoke[1] signatury. Např. API funkce `SetWindowsHookEx` má tvar

```
HHOOK WINAPI SetWindowsHookEx(
     __in int idHook,
     __in HOOKPROC lpfn,
     __in HINSTANCE hMod,
     __in DWORD dwThreadId);
```

[1] PInvoke = Platform Invoke, způsob použití unmanaged API funkcí v managed jazycích, jako např. C#nebo VB.NET

V PInvoke signatuře vypadá takto:

```
[DllImport("user32.dll", CharSet = CharSet.Unicode)]
static extern IntPtr SetWindowsHookEx(
     int idHook,
     KeyboardHookProc callback,
     IntPtr hInstance,
     uint threadId
);
```

Všechny tyto signatury se dají najít na webu `http://pinvoke.net`.

V konstruktoru této třídy je zavolána funkce `SetWindowsHookEx` (viz kap. 3.2.2) s typem háku `WH_KEYBOARD_LL` a ukazatelem na hákovací proceduru. Dále třída `GlobalKeyboardHook` generuje tři události, na které lze v okenní aplikaci reagovat – `KeyDown`, `KeyPress` a `KeyUp`. Obsahuje také dvě veřejné metody:

- `GetActiveAppTitle`,
- `GetActiveAppName`.

Metoda `GetActiveAppTitle` vrací název okna na popředí (využití API funkcí `GetForegroundWindow` a `GetWindowText`). Tato metoda je použita pro indikaci změny sekce, viz kap. 4.

Metoda `GetActiveAppName` vrací název aplikace okna na popředí. Tato funkce využívá vestavěnou třídu `System.Diagnostics.Process` a její statickou metodu `GetProcessById`. Této metodě je předán jako parametr id procesu, ze kterého chceme název aplikace získat. Id procesu získáme zavoláním API funkce `GetWindowThreadProcessId` knihovny `user32.dll`, které je předán jako parametr handle okna na popředí (`GetForegroundWindow`). Název aplikace představuje vlastnost `MainModule.FileVersionInfo.FileDescription` získaného procesu (třída `System.Diagnostics.Process`).

Hákovací procedura, která zajišťuje funkcionalitu zachycování klávesnice, má název `HookProc`. Nejprve se v této proceduře zkontroluje, zda její parametr odpovídá některé ze zpráv `WM_KEYDOWN`, `WM_SYSKEYDOWN`, `WM_KEYUP` nebo `WM_SYSKEYUP`. Poté je zavolána API funkce `GetKeyboardState`. Ta získá aktuální stav všech virtuálních kláves[2]. Dále je zavolána funkce `ToUnicodeEx` s odpovídajícími parametry, pomocí které získáme hodnotu stisknuté klávesy. Vzhledem k tomu, že stisk klávesy při různých rozloženích klávesnice (anglické, české) může odpovídat různým znakům, je jedním z parametrů návratová hodnota API funkce `GetKeyboardLayout`, která vrací aktuální rozložení klávesnice. Nakonec se zkontroluje, zda je stisknuta klávesa SHIFT nebo zapnutý CAPS LOCK a v kladném případě je převeden znak na velký.

Nyní už máme získaný znak odpovídající stisknuté klávese a postupně jsou generovány události `KeyDown`, `KeyPress` a `KeyUp`.

5.3 Modul KeyboardSkillsAnalysis

Tento modul představuje spustitelnou okenní aplikaci, která zobrazuje vyhodnocené výsledky psaní uživatele. K zachycování kláves využívá modul `HookService` (viz kap. 5.2).

Hlavní obrazovku (GUI) aplikace představuje třída `DTMainView`. Zobrazuje všechny grafické prvky aplikace a plní je daty z databáze.

[2] virtuální klávesa = klávesa, která přímo negeneruje znak, ale používá se v kombinaci s jinými klávesami (např. SHIFT, ALT, CTRL, atd.)

Tato třída obstarává základní úkony jako načítání (při spuštění aplikace) a ukládání (při ukončování aplikace) nastavení. Pro práci s perzistentním nastavením aplikace je využito *Properties*, které je uloženo ve formě XML ve stejném adresáři jako spouštěná aplikace. Výhodou je jednoduchá práce díky vestavěným funkcím .NET.

Dále tato třída zajišťuje minimalizaci do tray lišty a obnovení po dvojkliku na ikonu aplikace.

5.3.1 Zpracování napsaného textu

Sekci popsanou v kap. 4 představuje třída `DTSection`. Ta obsahuje potřebné metody pro práci se sekcí (přidání znaku, odstranění znaku, získání informací o sekci, ...) a vlastnosti, které ji charakterizují:

- Název aplikace – třída `string`.
- Název titulku okna – třída `string`.
- Čas psaní – třída `DTSectionTime`, která obsahuje složky počáteční a koncový čas psaní.
- Obsažený text – třída `DTText`. Obsahuje text sekce a metody pro získání prvních dvou slov textu, posledních dvou slov atd.
- Počet slov – int.
- Překlepové chyby – třída `DTMistakes`.
- Rytmičnost psaní – třída `DTRytmicity`.
- Počet úhozů za minutu – třída `DTKeystrokes`. Obsahuje počet zaznamenaných stisků kláves, body za stisknuté znaky a údaj o přesnosti psaní. Výsledná hodnota je získána postupem popsaným v kapitole 4.5.

Zpracování překlepových chyb

Vyhodnocování překlepových chyb je zpracováno ve třídě `DTMistakes`. Tato třída obsahuje seznam chyb (třída `DTMistake` – obsahuje informaci o chybném znaku, opravném znaku a slově, ve kterém k chybě došlo), seznam chybných znaků (implementováno pomocí generického typu `System.Collections.Generic.List`) a seznam s opravnými znaky (stejný typ jako chybné znaky). Při přidávání chybného znaku metodou `AddMistakeChar` se přidá znak do seznamu chyb, stejně tak řeší případ přidávání opravných znaků metoda `AddRepairChar`. Poté, co je v sekci dokončeno slovo, probíhá hledání chybného znaku podle algoritmu popsaného v kap. 4.3.

Zpracování rytmičnosti psaní

O zpracování rytmičnosti psaní se stará třída `DTRytmicity`. Obsahuje seznam písmen typu `DTLetterInfo` – ke každému znaku je uložen počet milisekund potřebných ke stisku odpovídající klávesy. Mimo tento seznam jsou údaje po každém stisku klávesy ukládány i do řetězce, který má formát `pismeno1:cas1;pismeno2:cas2;` atd. pro každé písmeno v sekci. Tato skutečnost souvisí s optimalizací ukládání těchto informací do databáze (viz následující podkapitola 5.3.2).

K měření uběhnutého času jsou použity stopky (třída `Stopwatch` ze `System.Diagnostics`). Po přidání znaku do seznamu a do řetězce jsou stopky vyrestartovány, aby měřily čas následujícího znaku.

5.3.2 Ukládání zpracovaných informací

Členem třídy `DTMainView` je i třída `DTKeylogger`. Ta vytváří instanci třídy `GlobalKeyboardHook` a stará se tedy o zachycování a zpracování stisků kláves tím, že obstarává události `KeyDown`, `KeyPress` a `KeyUp` této třídy.

V těchto metodách se realizuje spouštění a zastavování časovače pro měření nečinnosti uživatele (vestavěná třída `Timer` ze `System.Windows.Forms`). Jako optimální hodnota pro interval časovače se po testování jeví 5 s. Po uplynutí této doby od posledního stisku klávesy (po každém stisku se časovač resetuje) tedy dojde k ukončení sekce (z důvodu nečinnosti uživatele, viz kap. 4.1). Tato hodnota ovlivňuje i maximální dobu potřebnou ke stisku znaku – pokud tedy bude interval mezi stisky kláves trvat pod 5 s, bude se jednat o dobu stisku, pokud bude nad tuto mez, dojde k ukončení sekce a stisk dalšího znaku bude znamenat začátek nové sekce.

V metodě `KeyPress` (tedy po každém stisku znaku) je zavolána metoda `AddChar` třídy `DTSection` s parametrem stisknutého znaku a příznakem, zda došlo ke změně sekce.

Metoda `AddChar` má několik úloh:

- počítat slova v sekci – zvýšení počtu v případě, že nově přidaný znak je bílý znak (využití funkce `System.Char.IsSeparator`) a předchozí znak je písmeno, číslice (`Char.IsLetterOrDigit`) nebo interpunkční znaménko (`Char.IsPunctuation`) nebo nastala změna sekce,
- zaznamenávat počáteční čas sekce – pouze v případě, že se jedná o první znak v sekci,
- zaznamenávat koncový čas sekce – tato hodnota se přepisuje při každém stisku klávesy,
- předat informaci o aktuálním znaku třídě `DTKeystrokes` – třída zpracovává vyhodnocení počtu úhozů.

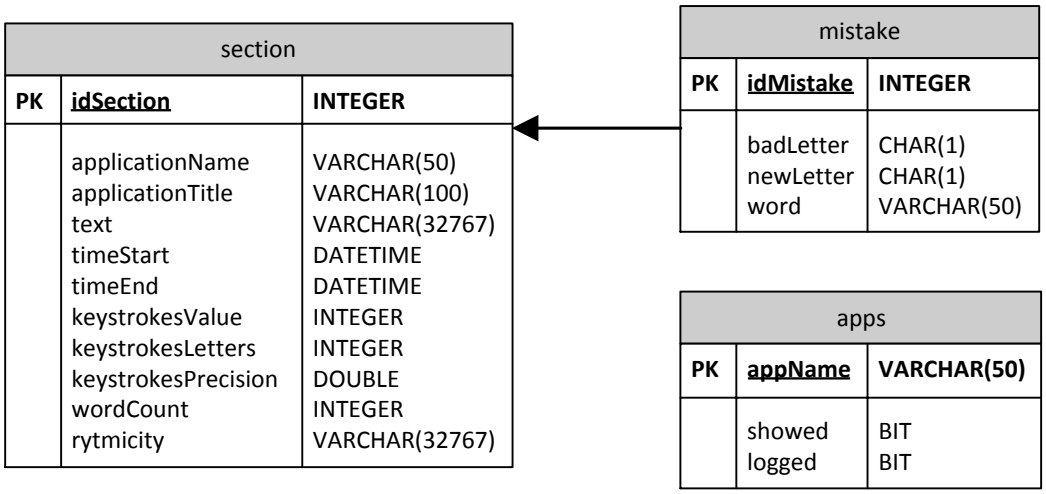
Další funkcí této třídy je kontrola změny sekce. Detekce změny (přepnutí aktivního okna) je zajištěna voláním funkce `GetActiveAppTitle` (viz kap. 5.2) v metodě `KeyDown`. Probíhá zde také kontrola, zda je název aktivního okna shodný s předešlým (viz kap. 4).

Schéma databáze

Pokud dojde k ukončení sekce z důvodu nečinnosti uživatele nebo změny programu, je sekce uložena do databáze (pokud je v naší aplikaci nastaveno povolené logování aplikace, ke které daná sekce patří). Její schéma je na obr. 5.1.

Vidíme, že pro ukládání dat naší aplikace jsou použity pouze 3 tabulky:

- section,
- apps,
- mistake.



Obrázek 5.1: Schéma databáze aplikace.

Tabulka *section* odpovídá sekci (viz kap. 4.1) – třídě `DTSection`. Pouze rytmičnost psaní (časy potřebné ke stisku kláves) je uložena jako řetězec. Původně byla pro rytmičnost vytvořena vlastní tabulka navázána na tabulku section, která měla 2 sloupce – písmeno a čas. Toto řešení ale mělo nevýhodu v tom, že po skončení sekce se do této tabulky ukládaly záznamy pro každé stisknuté písmeno v sekci. Při velkém počtu záznamů bylo znatelné zpomalení počítače. Proto je rytmičnost uložena jako řetězec popsaný v kap. 5.3.1, který je tvořen průběžně po stisku každého znaku. Zpracování tedy není nárazové jako v případě zvláštní tabulky a je rychlejší. Tento způsob má ale na druhou stranu za následek pomalejší načtení dat, protože řetězec musíme programově rozparsovat. Vzhledem k tomu, že ale jde o mnohem méně častou činnost než ukládání dat do databáze (ukládání při každé změně sekce, načítání pouze pokud si prohlížíme vyhodnocené výsledky), je toto vhodné řešení a zpomalení navíc není nijak znatelné.

Tabulka *mistake* je navázána na tabulku section a představuje seznam chyb v této sekci. Její struktura odpovídá třídě `DTMistake` (5.3.1).

Tabulka *apps* obsahuje seznam všech namonitorovaných aplikací a nastavení o jejich filtrování, tedy příznaky, zda má být aplikace zobrazována a zda má být logována. Vidíme, že mezi tabulkami *section* a *apps* není žádný vztah, ačkoli dobrý návrh databáze by vypadal tak, že název aplikací by byl pouze v tabulce *apps* a tabulka *section* by obsahovala cizí klíč z tabulky *apps*, který by aplikaci identifikoval. To by ale vyžadovalo vyhledávání aplikace v tabulce *apps* při každém vkládání nového záznamu a spojení těchto dvou tabulek pomocí `JOIN` při každém načítání dat. Vzhledem k tomu, jak je ukládání a načítání dat implementováno (viz 5.3.3), je toto řešení vhodnější a vede k rychlejšímu zpracování výsledků (zvláště při načítání velkého množství dat).

5.3.3 Práce s databází

Pro práci s databází je použito datových adaptérů, jež jsou implementovány pro všechny tabulky. Pro každý dotaz je vytvořena metoda adaptéru, která tento dotaz provede. My

potom přistupujeme k adaptérům jako k běžným objektům a veškerou práci nad databází provádíme skrze tyto metody. Všechny operace nad adaptéry zapouzdřuje singleton třída `DatabaseDataSet`.

Ukládání dat

Vložení nové sekce do databáze probíhá v metodě `InsertSection` s parametrem `DTSection` třídy `DatabaseDataSet`. Třída představující adaptér, který obstarává operace nad tabulkou *section*, má název `sectionTableAdapter`. Nejprve je zavolána metoda této třídy, která vloží do tabulky novou sekci– `InsertSection`. Tato metoda vykoná obyčejný insert dotaz:

```
INSERT INTO section (applicationName, applicationTitle, text, timeStart,
timeEnd, keystrokesValue, keystrokesLetters, wordCount, rytmicity)
VALUES (@applicationName, @applicationTitle, @text, @timeStart, @timeEnd,
@keystrokesValue, @keystrokesLetters, @wordCount, @rytmicity).
```

Hodnoty sloupců jsou předány jako parametry dotazu. Abychom získali id nově vložené sekce, přidáme za tento dotaz navíc

```
SELECT SCOPE_IDENTITY().
```

Poté jsou do databáze uloženy všechny chyby. V cyklu se prochází seznam chyb uložený ve třídě `DTMistakes` a jsou přidávány nové záznamy do tabulky *mistake* pomocí adaptéru `mistakeTableAdapter` a jeho metody `Insert`, která vykoná obyčejný insert dotaz.

Nakonec přichází na řadu filtrování. O tabulku *apps*, která obsahuje nastavení pro filtrování aplikací, se stará třída adaptéru s názvem `appsTableAdapter`. Nejprve je zkontrolováno, zda je název aplikace nově vkládané sekce již v databázi obsažen. Metoda `Exists` vrací počet záznamů požadované aplikace. Dotaz, který je proveden, má podobu

```
SELECT COUNT(*) FROM apps WHERE (appName = @appName).
```

Pokud je tedy výsledek této metody roven nule (záznam pro aplikaci neexistuje), je zavolána další metoda tohoto adaptéru `InsertApp`, která insertovým dotazem přidá novou aplikaci do databáze s výchozími hodnotami – logování aplikace i zobrazení aplikace povoleno.

Načítání dat

Filtrované aplikace obstarává třída `appsTableAdapter`. Při spuštění naší aplikace jsou z tabulky *apps* získaná všechna data metodou `GetData`, která vykoná dotaz

```
SELECT appName, showed, logged FROM apps.
```

Vrácená hodnota této metody je třída `appsDataTable` představující tabulku *apps*. Pomocí dotazovacího jazyka LINQ ([6]) je zavolán dotaz `ToDictionary` na data této třídy, kde prvním parametrem je název aplikace a druhým řádek této tabulky. Tento dotaz vrátí generický typ `Dictionary` (slovník) ze `System.Collections.Generic`, jehož klíčem je právě název aplikace a hodnotou řádek tabulky. Tento slovník je používán k získávání informací o filtrování aplikací. Výhoda spočívá v rychlém přístupu ke slovníku pomocí

klíče v případě čtení hodnoty i zápisu. Zároveň není nutné dotazování databáze při každé kontrole, zda má aplikace povolené logování či zobrazení.

Klíčovým bodem načítání dat je načítání sekcí. To probíhá v metodě `GetSectionOverview`, která vrací přehled informací popsaný v kap. 4.6, tedy třídu `DTSectionOverview`. Tabulka obsahující požadované sekce (třída `sectionDataTable`) nasnímané od konkrétního počátečního data po koncové datum je získána metodou `GetSections` adaptéru `sectionTableAdapter`. Tato metoda vykoná dotaz:

```
SELECT applicationName, applicationTitle, idSection, keystrokesLetters,
keystrokesValue, rytmicity, text, timeEnd, timeStart, wordCount FROM
section WHERE (timeStart BETWEEN @timeStart AND @timeEnd).
```

Protože se v podmínce vyskytují dva sloupce, podle kterých hledáme sekce, jsou na ně přiděleny indexy, aby byl dotaz co nejrychlejší. Procházením získané tabulky cyklem `foreach` získáme všechny řádky tabulky. Aby byla sekce (konkrétní řádek) přidána do výsledného přehledu, musí splňovat následující podmínky:

- musí obsahovat víc jak 10 písmen,
- musí obsahovat víc jak 3 slova,
- musí být povolené zobrazení aplikace,
- musí být povolené logování aplikace.

Sekce nesplňující tyto podmínky nejsou do vyhodnocování zařazeny, neboť nemají velkou vypovídací hodnotu, nebo si uživatel nepřeje je vyhodnocovat.

Všechny ostatní sekce jsou vyhodnoceny a je nutné pro každou sekci získat seznam chyb. Tabulku s chybami navázanými na sekci získáme díky metodě `GetMistakesById`, které předáme jako parametr id aktuální sekce. Seznam chyb (`Collections.Generic.List`) s prvky typu `DTMistake` získáme z této tabulky, pokud použijeme LINQ dotaz `Select` a jako parametr použijeme λ-kalkul, který z řádku tabulky vytvoří třídu `DTMistake` a na výsledek následně použijeme další LINQ dotaz `ToList`:
`mistakesTable.Select(mistakeRow => new DTMistake(mistakeRow.badLetter[0], mistakeRow.newLetter[0], mistakeRow.word)).ToList()`.

5.3.4 Slučování sekcí

Po inicializování potřebných proměnných je vytvořena nová instance třídy `DTSection`. Do konstruktoru jsou předány všechny potřebné parametry, ze kterých sekce nastaví všechny své proměnné popsané v kap. 5.3.1. Poté už je zavolána metoda `AddSection` třídy `DTSectionOverview`. Tato třída provádí slučování sekcí popsané v kap. 4.6. Její bázová třída je `DTSection`. Obsahuje tedy všechny vlastnosti jako sekce, ale navíc má slovník s klíčem typu `String`, který představuje **název aplikace**, podle které probíhá slučování. Hodnota slovníku typu `DTMergedApplication` odpovídá aplikaci popsané v kap. 4.6. V metodě `AddSection` probíhá následující: pokud slovník neobsahuje klíč s názvem aplikace předávané sekce, je do tohoto slovníku sekce přidána; pokud však aplikace s takovým názvem existuje, je podle klíče nalezena odpovídající sekce-aplikace (`DTMergedApplication`) a je zavolána její metoda `Add`, která sloučí původní a novou sekci do jedné (4.6).

Třída `DTMergedApplication` je velmi podobná třídě `DTSectionOverview`. Její rodičovská třída je také `DTSection`. V metodě `Add` probíhá podobný proces jako v metodě

`AddSection` třídy `DTSectionOverview`. Stejně tak obsahuje slovník se sekcemi. Rozdíl je v tom, že slovník má jako klíč typ `string` představující **název dokumentu** a hodnotu typu `DTMergedDocument`.

Třída `DTMergedDocument` se od předchozích dvou mírně liší. Také dědí z třídy `DTSection`, ale neobsahuje slovník, nýbrž seznam s hodnotami `DTSection`. V metodě `Add` jsou potom předávané sekce vkládány do tohoto seznamu.

5.4 Výsledná aplikace

Mým cílem bylo vytvořit aplikaci, která bude uživatelsky přívětivá a intuitivní. Všechny prvky reagují korektně na zvětšování a zmenšování velikosti hlavního okna. Pro rozmístění prvků a pro automatickou změnu jejich velikosti při roztahování okna je využito *panelů* a nastavení potřebných vlastností jednotlivým prvkům – `Anchor` a `Dock`. Hlavní okno (třída `DTMainView`) je logicky členěno do dvou částí, které jsou odděleny pomocí *splitteru* (třída `SplitContainer` ze `System.Windows.Forms`). Díky tomu si uživatel může měnit poměr mezi oběma částmi tak, aby velikosti oken byly vyhovující.

Levá část obsahuje dvě komponenty (třída `DateTimePicker`) pro výběr počátečního a koncového data a strom zobrazující hierarchii načtených aplikací a dokumentů (viz 5.4.1). Zvolená data určují, z jakého rozmezí budou brána data k vyhodnocení výsledků našich psacích schopností. Po potvrzení kliknutím na zelenou „fajfku“ je zavolána metoda `GetSectionOverview` (viz 5.3.3) třídy `DatabaseDataSet`, která načte všechna data z tohoto rozmezí. Následně jsou funkcí `UpdateView` třídy `DTMainView` zobrazeny všechny vyhodnocené výsledky.

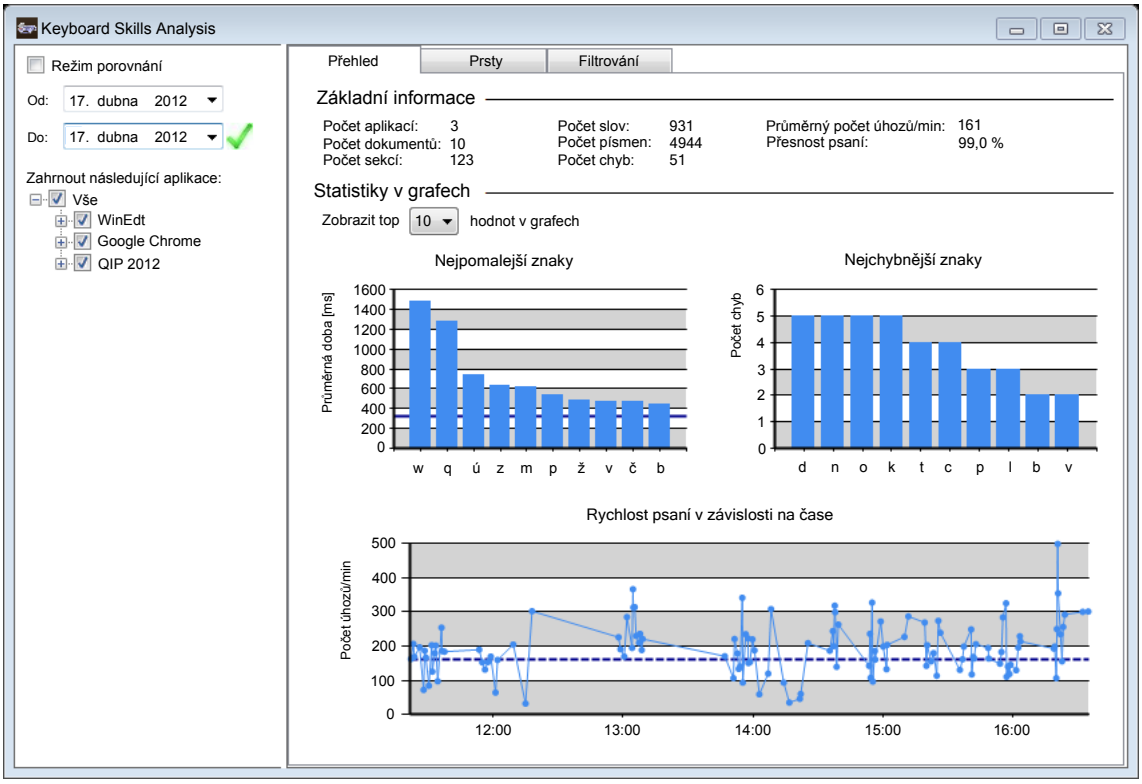
V této části je dále *checkbox*, který umožňuje zapnout *režim porovnání*. V tomto režimu jsou zobrazeny komponenty pro výběr dat a strom aplikací dvakrát. Je tedy umožněno spolu porovnávat 2 časové úseky, přičemž každý z nich může být jinak dlouhý. Můžeme např. zjistit, jak se naše schopnosti zlepšily oproti minulému měsíci, roku, atd. Výhodou je, že vidíme výsledky z obou časových úseků zároveň.

V databázi je u každé sekce uloženo jak datum vzniku, tak i čas. Hodnota získaná z komponenty pro výběr data představuje třídu `DateTime` obsahující složky pro datum i čas. Získaný datum odpovídá datu zvoleném v komponentě. Získaný čas odpovídá aktuálnímu systémovému času. Pokud však jako počáteční datum zvolíme např. 1.1.2012, zajisté chceme vidět výsledky od půlnoci tohoto data. Stejně tak pro koncové datum vyžadujeme záznamy pořízené až do půlnoci. Proto před použitím těchto dat použijeme konstruktory třídy `DateTime`, které vytvoří požadované nové objekty. Pro počáteční datum použijeme tvar `new DateTime(old.Year, old.Month, old.Day, 0, 0, 0);`, kde `old` představuje původní instanci data. Předávané parametry konstruktoru jsou v pořadí: rok, měsíc, den, hodiny, minuty, sekundy a milisekundy. Pro koncový čas potom použijeme tvar `new DateTime(old.Year, old.Month, old.Day, 23, 59, 59, 999);`. To nám zajistí výběr všech požadovaných záznamů.

V pravé části se nachází prvek `TabControl` se třemi přepínacími záložkami. Vzledem k tomu, že strom aplikací a zvolené datum se vztahuje ke všem těmto záložkám, bylo použito právě *splitteru*, aby tyto komponenty zůstaly vždy zobrazeny.

První záložka zobrazuje základní informace a statistiky zobrazené v grafech (viz kap. 5.4.2). Výsledná aplikace se zvolenou první záložkou je na obr. 5.2.

Druhá záložka obsahuje údaje o průměrném čase potřebném ke stisku znaků podle jednotlivých prstů. U každého prstu je `Label`, který tuto hodnotu zobrazuje. Popiskům nelze jednoduše nastavit vlastnosti `Anchor` a `Dock`, aby se správně posouvaly v případě zvětšování



Obrázek 5.2: Ukázka výsledné aplikace. V levé části je strom s aplikacemi, ze kterých byla pořízena data 17. dubna 2012. V pravé části jsou zobrazeny základní vyhodnocené informace a statistiky v grafech.

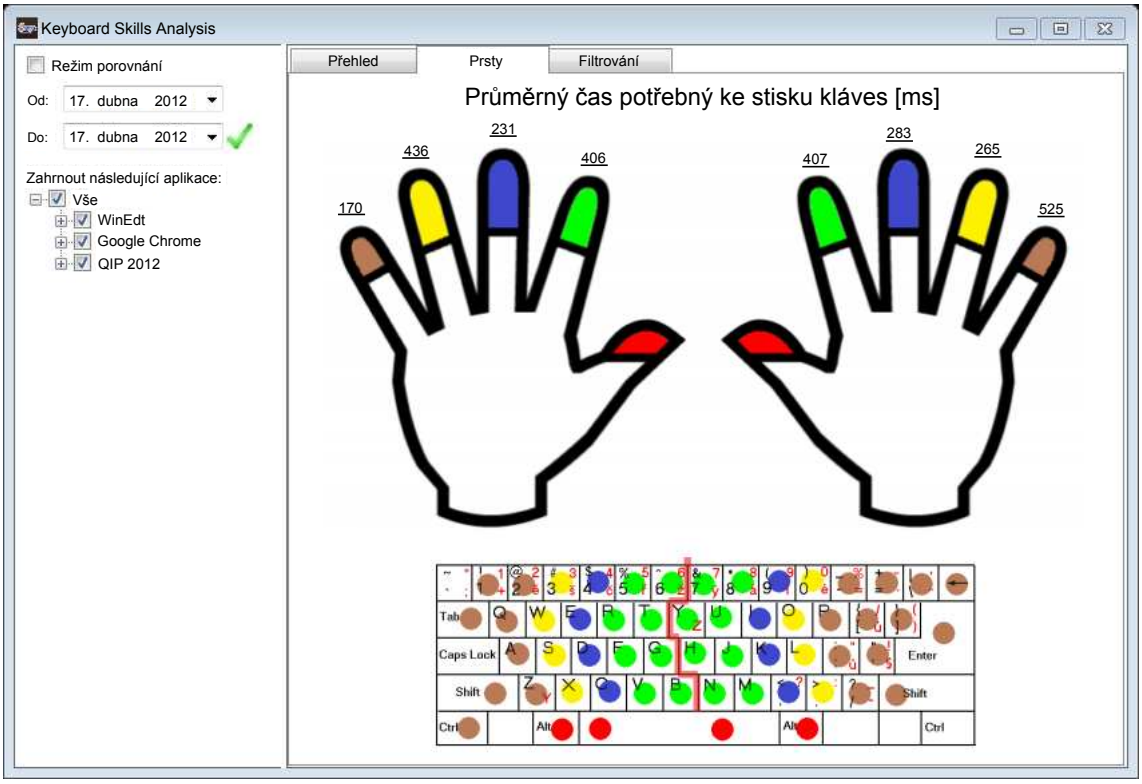
či zmenšování okna. Jejich pozice je tedy počítána podle poměru šířky a výšky příslušného obrázku. Po najetí na jednotlivé popisky myší je zobrazen graf, který ukazuje údaje o jednotlivých písmenech odpovídajícího prstu. K zobrazení těchto grafů je použita komponenta `DTToolTipFingers` zděděná z třídy `System.Windows.Forms.ToolTip`. Funkčnost implementuje metoda `SetToolTip`, ve které je dynamicky vytvořen objekt grafu `DTChartFingers` (dědí z třídy `System.Windows.Forms.DataVisualization.Charting.Chart`) uložený do paměti jako obrázek. O vykreslení se pak stará přepsaná virtuální metoda `OnDraw`. Náhled této záložky můžeme vidět na obr. 5.3.

Ve třetí záložce najdeme komponentu `DataGridView`. Ta slouží k nastavení filtrů nasnímaných aplikací (viz obr. 5.4). Uživatel zde může pomocí *checkboxů* buď zakázat vyhodnocování určitých aplikací či přímo zakázat logování těchto aplikací (viz kap. 4.2).

5.4.1 Strom aplikací a dokumentů

Pro zobrazení aplikací obsažených ve vyhodnocení jsem použil strom. Výhodou je možnost použít víceúrovňové zanoření tak, aby odpovídalo hierarchii popsané v kap. 4.6. Na nejvyšší úrovni je položka *Vše*, která odpovídá *přehledu*. O úroveň níž jsou zobrazeny všechny *aplikace*, které mají ještě jednu podúroveň. Ta odpovídá *dokumentům*. Poslední úroveň (*sekce*) už ve stromě zobrazena není, protože by neměla příliš velký význam.

Jednotlivým položkám ve stromě jsou přidány *checkboxy*, aby měl uživatel možnost dynamicky měnit, jaké aplikace či dokumenty chce ve vyhodnocení vidět. Po označení (od-



Obrázek 5.3: Ukázka výsledné aplikace se zobrazením času potřebného ke stisku kláves podle jenotlivých prstů. Po najetí na konkrétní hodnotu u prstu je zobrazen sloupcový graf, který zobrazuje hodnoty pro každý znak tohoto prstu.

značení) položky dojde ihned k přepočítání výsledků a aktualizují se číselné hodnoty i grafy. Ve třídě `DTTreeView` jsou implementovány funkce pro sestavení stromu ze seznamu aplikací (tříd `DTMergedApplication`, kap. 5.3.4) a dále rekurzivní funkce pro označení (odznačení) všech potomků při kliku na otcovskou položku. Díky tomu je možné např. odznačit všechny dokumenty stejné aplikace kliknutím na příslušnou položku aplikace.

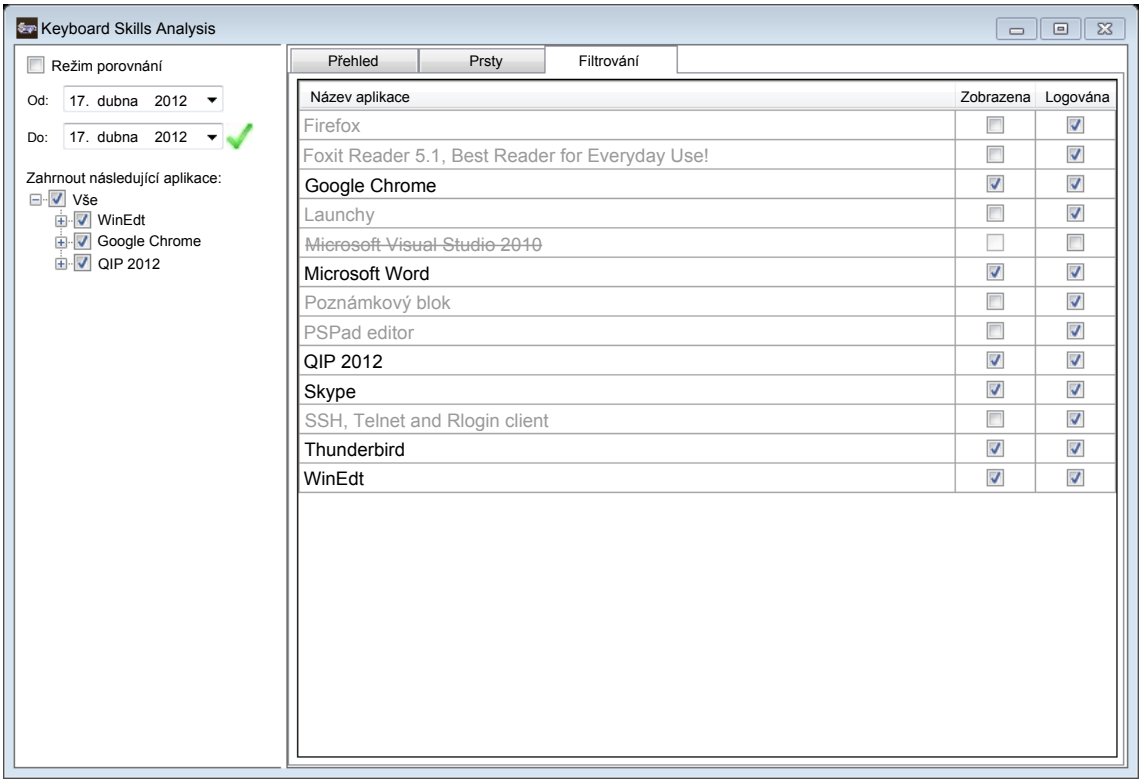
Odznačením položky dosáhne uživatel stejného efektu, jako kdyby nastavil aplikaci ve filtrech *zákaz zobrazení*. Rozdíl je v tom, že tato aplikace nebude zobrazena pouze dočasně a také toto nastavení nebude perzistentně uloženo.

5.4.2 Zobrazení grafů

Pro zobrazení grafů jsem zvolil vestavěnou třídu `Chart` ze jmenného prostoru `System.Windows.Forms.DataVisualization.Charting` (viz kap. 3.3.1). O zobrazení dat v grafech se stará metoda `ShowCharts` třídy `DTMainView`.

Aplikace ukazuje informace o schopnostech psaní ve 3 grafech:

- nejpomalejší znaky,
- nejchybnější znaky a
- rychlost psaní v závislosti na čase.



Obrázek 5.4: Ukázka nastavení filtrování aplikací. Aplikace, které si nepřejeme zobrazovat, jsou pro zpřehlednění zašedlé. Ty, které nechceme ani logovat, jsou navíc přeškrtnuté.

První dva grafy představují sloupcové grafy. Aby si mohl uživatel zobrazit vyhovující počet záznamů, přidal jsem nad tyto grafy `ComboBox`, kde si může hodnotu vybrat. Tato hodnota zůstává perzistentní i po ukončení aplikace, protože je ukládána do *properties* aplikace. V této souvislosti jsem narazil na problém, když chtěl uživatel vidět víc než 10 záznamů – nebyly zobrazeny popisky pro všechny sloupce, ale jen pro některé. Řešením bylo nastavit pro osu x vlastnost `Interval` na hodnotu 1 místo hodnoty `Auto`.

Nejpomalejší znaky

Abychom mohli zkoumat, která písmena nám dělají největší problémy, potřebujeme graf zobrazující znaky, u kterých byl průměrný čas potřebný ke stisku odpovídající klávesy nejdelší. K tomuto účelu se nejvíce hodí graf sloupcový, protože jsou vidět nejen poměry mezi znaky, ale i konkrétní hodnoty. Data, ze kterých je tvořen, pochází ze třídy `DTRytmicity`, kde je uložen seznam znaků a jejich doby stisku (viz kap. 5.3). Do tohoto seznamu jsou znaky ukládány průběžně při psaní textu. Před použitím těchto dat v grafu je zavolána metoda `PrepareToShow`, která tento seznam převede do slovníku (`Dictionary`), kde klíčem je znak a hodnotou počet milisekund. Při procházení seznamem se jednotlivé časy pro konkrétní znaky sčítají a nakonec je výsledný čas vydělen počtem těchto znaků – vypočítáme tedy aritmetický průměr. V *tooltipu* jednotlivých hodnot v grafu je navíc zobrazena přesná hodnota a počet výskytů tohoto znaku ve výpočtu. Může se totiž např. stát, že některý znak bude zobrazen jako nejpomalejší, ale přitom se v textu vyskytl v malém počtu. Větší pozornost ale musíme věnovat znaku, který je sice v průměru rychlejší, ale

v textu se vyskytuje mnohem častěji.

Nyní tedy máme slovník se znaky a průměrnou dobou stisku. Pokud ale chceme zobrazit pouze určitý počet záznamů (podle zvolené hodnoty v *comboboxu*), potřebujeme tento slovník seřadit, a to od největší hodnoty po nejmenší, a následně vzít pouze určitý počet prvních položek. Toho dosáhneme díky dotazovacímu jazyku LINQ, který použijeme hned 2krát za sebou. Nejprve na slovník použijeme dotaz `OrderByDescending` s parametrem λ-výrazu `x => x.Value` a následně dotaz `Take`, kde jako parametr předáme zvolenou hodnotu z *checkboxu*.

Přidávání konkrétních hodnot do grafu probíhá při průchodu slovníkem. Každý graf obsahuje kolekci sérií (pro případ, že bychom v jednom grafu chtěli zobrazit např. sloupcový a koláčový typ zároveň nebo více sloupců pro jednu hodnotu). V našem případě využijeme pouze jednu sérii (dvě, pokud máme zapnutý režim porovnávání). Tato série obsahuje kolekci bodů (`DataPointCollection`), do které přidáváme nové hodnoty metodou `Add`. Předávaný parametr je třída `DataPoint`. Té se předávají do konstruktoru 2 parametry: x-ová a y-ová hodnota. V tomto případě, kdy se jedná o sloupcový graf, použijeme jako x-ovou hodnotu jakékoli číslo (např. 0), ale pro všechny body stejné. Počet milisekund představuje y-ovou hodnotu.

V grafu je navíc zobrazena horizontální čára představující průměrnou dobu stisků všech znaků (ne pouze zobrazených).

Nejchybnější znaky

Informace o překlepech, kterých se dopouštíme, se dozvíme ze sloupcového grafu, který zobrazuje znaky, ve kterých se nejvíc chybuje. Využívá přitom data třídy `DTMistakes` (viz kap. 5.3), která obsahuje seznam chyb. Podobně jako u grafu s nejpomalejšími znaky, i zde je před použitím dat zavolána metoda `PrepareToShow` třídy `DTMistakes`, která uloží chyby do slovníku, kde klíčem je znak a hodnotou počet chyb v tomto znaku. Procházením seznamu se zvedá počet o 1 odpovídajícímu znaku.

Takto získaný slovník potřebujeme opět seřadit podle hodnot, a to od největší po nejmenší, a vzít jen určitý počet. Postupujeme stejným způsobem jako v případě grafu s nejpomalejšími znaky. Stejně tak přidávání konkrétních hodnot do grafu probíhá obdobným způsobem.

V *tooltipu* jednotlivých hodnot je zobrazen přesný počet chyb u daného znaku, dále pak celkový výskyt znaku v textu a jeho procentuální chybovost. Ta je získáná pomocí rovnice 5.1, kde f je výsledná chybovost v procentech, o je počet chyb a c je počet výskytů znaku.

$$f = \frac{o}{c} \cdot 100 \tag{5.1}$$

Rychlost psaní v závislosti na čase

V kap. 2.3 jsem se zmiňoval, že jedním z faktorů, které mohou ovlivňovat kvalitu psaní na klávesnici, je denní doba. Tuto závislost můžeme vyčíst ze spojnicového grafu, který má časovou osu x a na ose y vynesenou hodnotu počtu úhozů za minutu pro konkrétní časy. Z grafu můžeme vyčíst nejenom závislost na denní době, ale i např. vývoj psacích schopností v čase.

Data pro tento graf vznikají ve třídě `DTSectionOverview` při načítání dat z databáze. Do slovníku, kde klíčem je čas a hodnotou je sekce (`DTSection`), jsou uloženy veškeré načtené záznamy. Před použitím je však nutné tento seznam seřadit podle času (klíče)

od nejmenšího po největší. Použijeme tedy na slovník opět LINQ dotaz – `OrderBy(x => x.Key)`.

Přidávání dat do grafu probíhá podobně jako v případě obou sloupcových grafů s tím rozdílem, že je místo funkce `Add` použita funkce `AddXY`, která přijímá dva parametry – hodnotu pro osu x a hodnotu pro osu y. Prvním je čas a druhým je počet úhozů za minutu v tomto čase (v této sekci).

Protože x-ová souřadnice je časová, bylo nutné nastavit vlastnost `IntervalType` této osy na hodnotu `Minutes`, aby byla data správně zobrazena. Dále bylo nutné nastavit formát zobrazeného času na této ose. Ten je vytvořen následujícím způsobem. Formátovací řetězec (dále jen FŘ) inicializujeme na prázdný řetězec. Pokud je počáteční datum rozdílné od koncového, je do FŘ přidán tvar {`d. MMMM`}, který zobrazí den a měsíc. Dále pokud by byl rozdíl koncového a počátečního data menší než 7 dnů, přidáme řetězec {`HH:mm`}, který zobrazí i hodiny a minuty (ve stejný den tedy pouze hodiny a minuty). V případě, že je rozdíl koncového a počátečního data větší než 365 dnů, přidáme do FŘ {`yyyy`}, což zobrazí rok. Tím máme zajištěno, že na x-ové ose budou zobrazeny adekvátní údaje o čase vzhledem k rozsahu dat, který uživatel zvolil. Přesný čas uživatel zjistí také v případě, že najede myší na konkrétní bod v grafu – zobrazí se v *tooltipu*.

Čárkovaná horizontální čára zobrazuje průměrný počet úhozů za minutu ze všech sekcí.

5.5 Vyhodnocení systému

Vytvořenou aplikaci jsem pro účely testování předal několika uživatelům. Dostal jsem i několik připomínek k uživatelskému rozhraní. Tyto změny jsem do finální aplikace zapracoval, aby bylo GUI přehlednější a ovládání intuitivnější.

Vyhodnocené informace

Pro zjištění mých psacích schopností jsem vybral takové časové období, aby byly obsaženy všechny uložené záznamy v databázi (cca 3 měsíce, počet záznamů cca 2 200). Po odfiltrování nežádoucích aplikací (Firefox, Chrome, Visual Studio, ...) zůstaly ve vyhodnocení následující (všechny dokumenty v jednotlivých aplikacích ve stromě jsou označené):

- QIP,
- Thunderbird,
- Microsoft Word,
- Skype,
- WinEdt.

Základní vyhodnocení informací o psaní můžeme vidět v tabulce 5.1. Tyto hodnoty jasně ukazují, že mé psací schopnosti jsou na nízké úrovni – malá rychlost psaní a vysoká chybovost. Abychom se dozvěděli přesnější informace o tom, co zůsobuje největší problémy, podíváme se na další výsledky.

V tabulce 5.2 je zobrazeno 10 nejpomalejších znaků. V kapitole 2.3 jsem se zmiňoval o tom, že jedním z faktorů ovlivňujících rychlost psaní může být návyk na klávesy. První 4 znaky v této tabulce (*ú*, *w*, *x*, *q*) jsou skutečně znaky nejméně používané, proto jejich stisk trvá poměrně dlouho. Zajisté by bylo vhodné tyto znaky procvičovat a zlepšovat. Lepší

Počet aplikací	5
Počet dokumentů	22
Počet sekcí	615
Počet slov	5 861
Počet písmen	24 017
Počet chyb	922
Průměrný počet úhozů/min	131
Přesnost psaní	96,2 %

Tabulka 5.1: Základní vyhodnocené údaje ze zaznamenaných dat. Z původních cca 2 200 sekcí v databázi je vyhodnoceno pouze 614. To je způsobeno buďto odfiltrováním nežádoucích aplikací nebo nesplněním minimálního počtu slov a písmen v sekci.

však bude soustředit se na znaky jako *b*, *z*, *p*, *v*, *j*, neboť ty jsou v textu mnohem častěji používány.

Znak	**Průměrná doba stisku** [ms]	**Počet výskytů**
ú	1 143	25
w	1 051	14
x	916	32
q	892	6
b	745	318
z	640	375
p	609	763
v	535	663
j	480	541
č	478	164

Tabulka 5.2: 10 nejpomalejších znaků ve zvoleném období. Průměrná hodnota času potřebného ke stisku znaku je 331 ms. Všechny tyto znaky jsou tedy nad průměrem.

Rychlosti psaní podle konkrétních prstů jsou zobrazeny v tabulce 5.3. Jako nejrychlejší prst se jeví levý ukazováček, a to zřejmě proto, že obsluhuje buďto velmi často používané znaky (*a*, *y*) nebo velmi zřídka používaný znak *q*, který průměr příliš nezhorší. Naopak pravý malíček obsluhuje klávesy málo často používané, a proto je ze všech prstů nejpomalejší.

Seznam nejchybnějších znaků je zobrazen v tabulce 5.4. Zde nejsou znaky seřazeny podle procentuální chybovosti, ale podle celkového počtu chyb. Vidíme například, že chybovost u znaku *a* je 7,1 %. Ve znacích jako *l*, *d* nebo *u* se sice dopouštíme větší procentuální chybovosti, ale protože se v textu nevyskytují tak často, měli bychom se spíše zaměřit na ty znaky, které způsobí v konečném výsledku nejvíce chyb.

Z grafu rychlosti psaní v závislosti na čase je vidět lehké kolísání v průběhu 3 měsíců. Nedošlo ale k žádnému zpomalení ani zrychlení. Abychom něco takového mohli pozorovat, byl by pravděpodobně potřeba větší časový úsek.

Prst	**Průměrná doba stisku** [ms]	
	Levá ruka	Pravá ruka
Ukazováček	440	372
Prostředníček	248	278
Prsteníček	471	244
Malíček	178	598

Tabulka 5.3: Rychlosti stisků znaků podle prstů.

Znak	**Počet chyb**	**Počet výskytů**	**Chybovost** [%]
a	102	1 444	7,1
n	88	1 173	7,5
l	77	661	11,6
d	72	769	9,4
o	69	1 594	4,3
u	58	676	8,6
p	57	763	7,5
e	48	1 720	2,8
k	39	855	4,6
á	39	399	9,8

Tabulka 5.4: 10 nejchybnějších znaků vyskytujících se v textu ve zvoleném období.

Z předchozích informací je patrné, u kterých znaků vzniká nejvíce překlepů a které znaky trvá na klávesnici nejdéle „najít“. Díky těmto výsledkům vím, které klávesy procvičovat, abych se v psaní na klávesnici mohl zlepšovat.

Systémové nároky aplikace

Spuštěná aplikace téměř nevytěžuje procesor, využití kolísá mezi 0 a 1 %. Velikost spotřebované paměti záleží na tom, kolikrát si zobrazíme výsledky vyhodnocení. Při spuštění programu se tato hodnota pohybuje kolem 10 MB, což by na dnešních sestavách nemělo činit žádný problém.

Vyhodnocení výsledků v aplikaci je poměrně rychlé. Při výběru všech nasnímaných dat (cca 2200 sekcí v databázi) trvá výpočet méně než 1 s. Dynamický přepočet při označování (odznačování) aplikací a dokumentů ve stromě je téměř okamžitý.

Kapitola 6

Závěr

Cílem diplomové práce bylo prostudovat techniku hmatové metody pro rychlé psaní na klávesnici a dále možnosti zachycování kláves v systému Windows. Výsledná aplikace má umožnit sbírat data o psaní uživatele na klávesnici a vyhodnocovat kvalitu psaní a chybovost. Tento cíl se mi podařilo splnit. Nasbíral jsem několik dat od různých uživatelů a podle výsledků jsem upravoval použité algoritmy tak, aby bylo vyhodnocování co nejpřesnější.

Aplikace na pozadí snímá stisknuté klávesy, zaznamenává doby stisků znaků a rozděluje napsaný text do sekcí. Společně s tím detekuje překlepové chyby a ukládá všechny údaje do databáze. Z těchto záznamů je následně možné zobrazit vyhodnocené informace ve zvoleném časovém rozsahu. Údaje o nejpomalejších a nejchybnějších znacích zobrazuje přehledně v grafech. Tyto informace je možné zobrazit i pro konkrétní prsty. Dále je v aplikaci zobrazen ve formě grafu vývoj psaní v závislosti v čase. Možné je také filtrování aplikací, ze kterých si nepřejeme vyhodnocovat výsledky.

Provedl jsem vyhodnocení svých nasnímaných dat a zjistil jsem, že mé psací schopnosti jsou na nízké úrovni. Rozebral jsem jednotlivé části, z čehož vyplynuly nejpomalejší a nejchybnější znaky. Pro zlepšení psaní je tedy nutné zapracovat na procvičování těchto znaků.

Stejným způsobem může program sloužit ostatním uživatelům. Další možností je použít tento program jako doplněk pro aplikace sloužící k výuce psaní všemi deseti prsty.

V rámci dalšího rozvoje systému by bylo možné soustředit se na přidání detekce gramatických chyb. Jiným možným rozšířením je zobrazování dalších informací o psaní na klávesnici. Výhodou systému je možnost snadného přidání těchto komponent do již existující aplikace.

Práce na projektu byla velmi zajímavá. Dozvěděl jsem se mnoho informací o hmatové metodě a o správných zásadách psaní na klávesnici. Dále jsem získal vědomosti o možnostech zasahování do událostí ve Windows – hákování. Zajímavým poznatkem bylo obeznámení se s možnostmi tvorby grafických uživatelských rozhraní a grafů.

Literatura

[1] Microsoft: About Messages and Message Queues [online]. http://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx, [cit. 2012-01-20].

[2] Microsoft: Hooks Overview [online]. http://msdn.microsoft.com/en-us/library/windows/desktop/ms644959(v=VS.85).aspx, [cit. 2012-01-20].

[3] Neugebauer, T.: Psaní všemi deseti - hmatová metoda [online]. http://www.psani-vsemi-deseti.cz/2045/download/Psani_vsemi_deseti_hmatova_metoda.pdf, [cit. 2012-01-20].

[4] Petzold, C.: *Programování ve Windows*. Computer Press, 1999, 1216 s., iSBN 80-7226-206-8.

[5] Petzold, C.: *Programování Microsoft Windows Forms v jazyce C#*. Computer Press, 2006, 356 s., iSBN 80-251-1058-3.

[6] Pialorsi, P.; Russo, M.: *Microsoft LINQ*. Computer Press, 2009, 615 s., iSBN 978-80-251-2735-3.

[7] Rehr, D.: The First Typewriter [online]. http://home.earthlink.net/~dcrehr/firsttw.html, [cit. 2012-01-20].

[8] Rehr, D.: Why qwerty was invented [online]. http://home.earthlink.net/~dcrehr/whyqwert.html, [cit. 2012-01-20].

[9] Sells, C.: *C#a Winforms*. Zoner Press, 2005, 648 s., iSBN 80-86815-25-0.

[10] Sharp, J.: *Microsoft Visual C#2010*. Computer Press, 2010, 696 s., iSBN 978-80-251-3147-3.

[11] pmq SOFTWARE: Výpočet čisté rychlosti psaní [online]. http://www.vsemi-deseti.cz/1099/popis_vypocet.html, [cit. 2012-01-20].

[12] SWX: Optimální klávesnice [online]. http://www.swx.cz/index.php?pg=17, [cit. 2012-01-20].

Příloha A

Obsah DVD

Na DVD se nachází následující adresářová struktura:

- **KeyboardSkillsAnalysis** – projekt pro Visual Studio 2010. Obsahuje všechny zdrojové soubory aplikace i spustitelné soubory. Má další podsložky:
  - **HookService** – soubory modulu HookService a výsledný modul dll (ve složce **bin**)
  - **Keylogger** – soubory modulu KeyboardSkillsAnalysis a spustitelná aplikace (ve složce **bin**)
- **Plakat** – obsahuje plakát reprezentující tuto diplomovou práci (formát pdf i zdrojový soubor pro program Inkscape)
- **Zprava** – obsahuje zdrojové soubory této zprávy pro systém LaTeX, dokument této zprávy ve formátu pdf a všechny použité obrázky v této zprávě