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 PRE ANALÝZU PÍSANIA UŽÍVATEĽA NA KLÁVESNICI

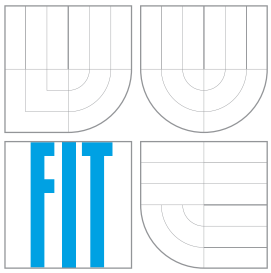
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ANTON GIERTLI

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 PRE ANALÝZU PÍSANIA UŽÍVATEĽA NA KLÁVESNICI

A TOOL FOR ANALYSIS OF USER'S TYPING SKILLS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ANTON GIERTLI
AUTHOR

VEDOUCÍ PRÁCE — Doc.,Ing. ADAM HEROUT, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Tato práce se věnuje analýze psaní uživatele na klávesnici. Důvodem jejího vzniku je snaha poskytnout uživateli komplexní pohled na jeho typografické schopnosti. Jejím výsledkem je aplikace kompatibilní s 32-bitovým operačním systémem Microsoft Windows, která na požádání poskytne uživateli statistiky týkající se jeho psaní. Patří mezi ně údaje o rychlosti psaní, chybovosti, překlepech, průměrné délce slov, histogram časových rozestupů mezi dvěme stlačenými klávesami, atd. Díky tomu může uživatel na svých schopnostech pracovat, sledovat jejich vývoj v čase a být motivován ke zlepšování. Práce se rovněž věnuje problémům spojeným s implementací této aplikace na platformě Microsoft Windows.

Abstract

This bachelor thesis is dedicated to analysis of user's typing. The reason of its origin is the aim to provide complex view on user's typographical skills. Its result is an application compatible with the 32-bit operating system Microsoft Windows, providing user with statistics when requested. These statistics include data about typing speed, error rate, typos, average word length, histogram of time differences between two keystrokes, etc. Thanks to this, user can work on his skills, watch their development in time and stay motivated for selfimproving. Thesis is also concerned with problems connected with implementations of application on Microsoft Windows platform.

Klíčová slova

psaní na klávesnici, analýza psaní, detekce překlepů, statistiky psaní

Keywords

typing, typing analysis, typos detection, typing statistics

Citace

Anton Giertli: Nástroj pre analýzu písania užívateľa
na klávesnici, bakalářská práce, Brno, FIT VUT v Brně, 2012

Nástroj pre analýzu písania užívateľa na klávesnici

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana docenta Adama Herouta

. . . . . .
Anton Giertli
10. května 2012

Poděkování

Ďakujem vedúcemu bakalárskej práce za znamenité vedenie počas oboch semestrov, ďalej priateľke, za konštruktívne pripomienky a nápady ohľadom grafickej podoby aplikácie, Fridolínovi Pokornému za trpezlivosť a dobré rady, a v neposlednom rade, všetkým priateľom a známym ktorí sa podieľali na testovaní aplikácie.

Obsah

Kapitola 1

Úvod

Písanie na počítači sa stalo každodennou súčasťou života, častokrát má dokonca vplyv na náš pracovný výkon, no málo ľudí k nemu pristupuje systematicky. V obehu je síce mnoho výukových aplikácií, ktoré sa zaoberajú práve cvičením správnej techniky písania, napríklad ATF[1], no chýbajú aplikácie, ktoré by poskytovali užívateľovi pohľad na jeho typografické schopnosti aj po skončení procesu učenia. Každý užívateľ, či už začiatočník, alebo pokročilý, má priestor na zlepšovanie vlastných typografických schopností. A práve tento priestor mu poskytuje aplikácia, ktorá je výsledkom tejto bakalárskej práce.

Cieľom práce je teda poskytnúť užívateľovi čo možno najužitočnejšie informácie o tom ako kvalitne, rýchle, alebo chybovo píše. To všetko vhodnou formou tak, aby mali výstupné informácie vysokú výpovednú hodnotu a k ich interpretácii nebolo treba vysokú úroveň počítačových znalostí. Dôraz je teda kladený na užívateľskú prívetivosť, nízku záťaž počítača (predpokladá sa, že aplikácia bude bežať na pozadí operačného systému dlhú dobu, je neprijateľné aby počítač neprimeraným spôsobom zaťažovala), kvalitu výstupných informácií.

Táto bakalárska práca sa zaoberá vývojom vyššie opisovanej aplikácie. Kapitola 2 je venovaná užívateľskému písaniu, kde patria známe techniky písania a skúmané charakteristiky písania. Kapitola 3 sa zaoberá odchytávaním kláves na platforme Microsoft Windows, je v nej načrtnutý problém prekladu stlačenej klávesy. V kapitole 4 sa čitateľ bližšie zoznámi s návrhom samotnej aplikácie, metodikou prístupu, rozložením problému na podproblémy. Piata kapitola sa venuje konkrétnej implementácii aplikácie pre platformu Windows, sú uvedené najzaujímavejšie pasáže implementácie. Dôležitou súčasťou implementácie bolo testovanie a dotazníky na základe ktorých vznikala finálna podoba aplikácie. Ich výsledky sú prezentované taktiež v tejto kapitole. V šiestej, poslednej kapitole, je zhrnutý prínos práce. Takisto je v nej navrhnuté možné pokračovania vývoja aplikácie.

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

Kapitola 2

Písanie na klávesnici

Táto kapitola je venovaná písaniu na klávesnici. Budú tu prezentované základné techniky písania, ich charakteristiky, spoločné ale aj rozdielne črty. Ďalej budú detailne rozobrané vybrané charakteristiky písania, ktoré som sa rozhodol skúmať. Naštudovanie týchto poznatkov je nutné k správnemu pochopeniu užívateľovho písania na klávesnici a pomáha pochopiť túto činnosť. Nasledujúce informácie teda poslúžili ako základ pri návrhu aplikácie a interpretácii výsledkov, ktoré boli aplikáciou získané.

2.1 Techniky písania

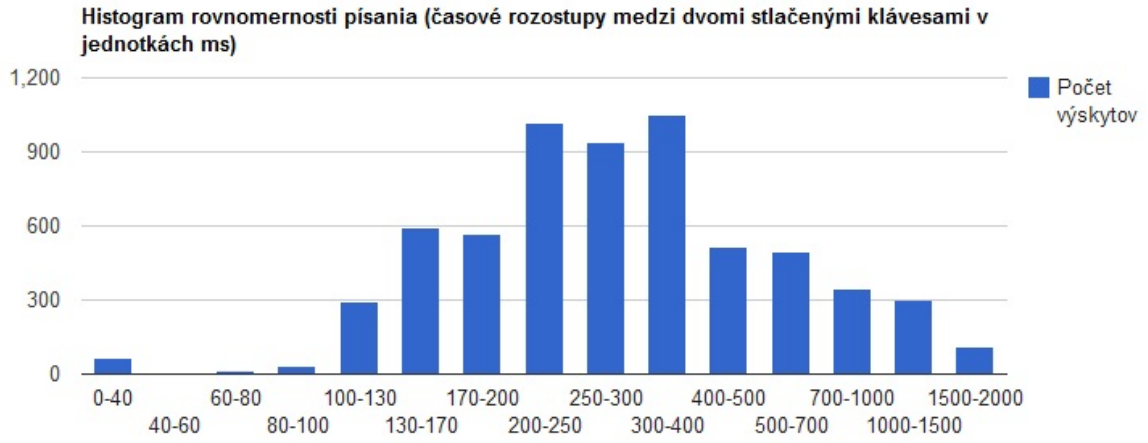
Táto kapitola sa zaoberá technikami písania a ich vzájomným porovnaním. Pochopenie týchto techník pomáha napríklad nájsť aspekty skúmania užívateľovho písania. Ďalej, techniky písania určujú rozdelenie užívateľov podľa rýchlosti písania. Je napríklad všeobecne známe, že hmatová metóda je spomedzi všetkých najrýchlejšia[7]. Častokrát technika písania súvisí s rovnomernosťou písania. Na základe týchto informácií, je možné užívateľovi podať informáciu na akej úrovni sa jeho písanie pohybuje.

2.1.1 Technika "ďobania"[1]

Túto techniku používajú najmä začiatočníci pri práci s počítačmi. Užívateľ pri nej zvyčajne používa len dva prsty a pred každou napísanou klávesou sa musí pozrieť na klávesnicu a nájsť požadovanú klávesu. Táto metóda je zväčša pomalšia ako hmatová metóda, ktorá bude rozoberaná nižšie v texte. Jedným z dôvodov nižšej rýchlosti je fakt, že užívateľ pri písaní používa menej prstov, a teda ich musí presúvať vo väčších vzdialenostiach. Hoci touto metódou môže byť dosiahnutá relatívne vysoká presnosť, je vysoká šanca, že užívateľ si chybu, ktorú spraví nevšimne, práve z dôvodu neustáleho pozerania sa na klávesnicu namiesto na obrazovku. Existuje mnoho odvodenín tejto techniky, ktoré by boli niekde na polceste medzi technickou ďobania a hmatovou technikou. Napríklad užívateľ sa síce nepozerá na klávesnicu, no stále píše len dvomi prstami, často krát nesystematicky. Technika má však opodstatnené využitie pri písaní na klávesnici s neznámym alebo novým rozložením, napríklad pri prechode zo štandardnej QWERTY klávesnice na zjednodušenú Dvorakovu klávesnicu[8, str. 18-25].

[1] z anglického Hunt and Peck

Z vlastností metódy ďobania vyplýva, že užívateľ, ktorý túto techniku používa, píše pomalšie aj nepravidelnejšie. Toto tvrdenie je podložené nasledovným histogramom časových rozostupov medzi dvomi stlačenými klávesami.



Obr. 2.1: Histogram užívateľa, ktorý píše dvomi prstami

Jasne na ňom vidieť, že rozostupy medzi dvomi stlačenými klávesami sú relatívne vysoké a navyše vysoké hodnoty je badať vo veľkom počte intervalov. Takýto histogram zväčša majú užívatelia, ktorí nie sú veľmi zbehlí v práci na počítači, a aplikácia ich na ich nedostatky patrične upozorňuje.

2.1.2 Hmatová technika

Prvé zmienky o hmatovej metóde písania sú vyše 100 rokov staré. V roku 1890 Lovisa Ellen Bullard Bernes definovala hmatovú metódu nasledovne[2]:

> Naučiť sa písať hmatom znamená pozerať sa na klávesnicu iba zriedka, sedieť priamo pred prístrojom. Mať ruky čo najbližšie pri sebe v jednej pozícii nad klávesnicou.

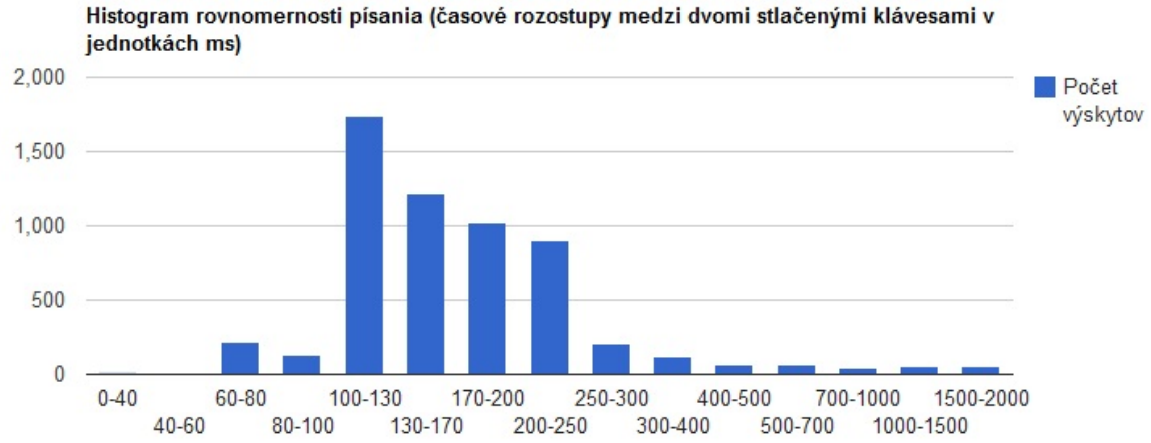
Princíp hmatovej metódy spočíva v zautomatizovaní písania slov, či slovných spojení, a to tak, že sa už nebude jednať o vedomú ale o podvedomú činnosť. Dôsledkom tohto je, že užívateľ má možnosť viacej sa venovať vlastným myšlienkam ako premýšľaním nad samotnou komunikáciou medzi ním a počítačom. Písanie sa stáva plynulejším a frekvencia chýb je nižšia. A to všetko bez toho, aby sa užívateľ musel na obrazovku pozerať. Samozrejme, medzi nevýhody patrí relatívne zdĺhavý proces učenia - minimálne 70 hodín [6]. Výhod je však omnoho viacej, patrí medzi ne napríklad:

- vyššia rýchlosť generovania znakov
- menšia chybovosť
- automatická kontrola správnosti zápisu, ktorá umožňuje opravu prípadnej chyby priamo pri písani, pretože po dostredivom nervovom vlákne ide do nervového centra nielen

prijatý vzruch, ale aj informácie o vykonanej úlohe (ak je chybná, dôjde k uvedomeniu si chyby bez kontroly generovaniu znakov)

- možnosť plynulého prepínania medzi jednotlivými rozloženiami klávesnice

Z popisu hmatovej metódy vyplýva, aj to, že užívateľ ktorý touto technikou píše by mal písať pravidelnejšie. Toto tvrdenie je podložené nasledovným histogramom, kde jasne vidieť, že rozostupy daného užívateľa medzi dvomi stlačenými klávesami sú nižšie, a vysoké hodnoty sú zastúpené len v niektorých intervaloch.



Obr. 2.2: Histogram užívateľa ovládajúceho hmatovú techniku písania

2.1.3 Ďalšie techniky písania

Existuje ešte niekoľko ďalších techník písania, ktoré síce možno nie sú až také rozšírené, ale určite stoja za zmienku. Prvou z nich je klasická technika písania desiatimi prstami. Nevýhodou tejto techniky je, že užívateľ sa stále musí pozerať na klávesnicu, preto sa zvyšuje riziko neodhalenia chyby. Priamym nasledovníkom tejto metódy je hmatová metóda. Ďalšou metódou je tzv. thumbing[2], táto metóda sa začala využívať s nástupom smartphonov, resp. mobilných telefónov s väčšou dotykovou obrazovkou. Ak by výsledná aplikácia tejto bakalárskej práce bola určená aj pre mobilné zariadenia, určite by bolo užitočné si túto metódu bližšie naštudovať.

2.2 Vybrané charakteristiky písania

Ako už bolo povedané, písanie je komplexná činnosť, ktorú možno definovať viacerými charakteristikami. Je treba teda tieto charakteristiky určiť, aby bolo jasné čo sa skúmať oplatí, a čo naopak je skúmať redundantné. Určite nemá zmysel skúmať každý aspekt písania, ak by nám nemal priniesť nejakú výpovednú hodnotu. Zameral som sa teda na také charakteristiky, ktoré sú jednoznačne vyčísliteľné. Nebudem teda skúmať či užívateľ píše poeticky alebo sofistikovane.

Dve pomocné charakteristiky sú **počet úderov** do klávesnice a **počet slov**. Tieto charakteristiky nám samy o sebe neprinášajú veľkú výpovednú hodnotu, no ich hodnoty sa

[2] palcovanie

využijú pri počítaní ďalších veličín. Počet úderov do klávesnice netreba ďalej rozoberať, výpočet tejto veličiny je celkom očividný zo samotné názvu a významu veličiny. Zastavím sa ale pri počte slov. Táto charakteristika je možno na prvý pohľad jednoznačná, no treba exaktne určiť detekciu nového slova. Začiatok nového slova je možné detekovať po stlačení týchto kláves - medzerník, enter, tabulátor. Treba brať do úvahy aj to, že užívateľ môže tieto klávesy stlačiť viac krát po sebe a túto postupnosť je nutné interpretovať ako jedno stlačenie.

Z vyššie uvedených veličín je možné spočítať priemernú dĺžku slova ako podiel počtu slov a počtu úderov.

Ďalšou dôležitou skúmanou veličinou je rýchlosť písania. Jej výpočet už nie je taký triviálny. Jednotka tejto veličiny je **počet slov za minútu**. Čo možno znie trošku zavádzajúco, lebo v samotnom vzorci pre výpočet tejto veličiny sa nikde počet slov nevyskytuje. Na základe empirických meraní sa zistilo[23, str. 175-202], že priemerná dĺžka slova je päť, a preto sa vo vzorci delí počet stlačených kláves (vrátane špeciálnych znakov ako je medzerník, enter, atď.) číslom 5. Finálny vzťah je teda nasledovný:

$$V = \frac{|T| - 1}{S} * 60 * \frac{1}{5}[1] \tag{2.1}$$

Kde S je čas od stlačenia prvej klávesy až do stlačenia poslednej klávesy. $|T|$ reprezentuje dĺžku zaznamenaného textu. Konštanta *60* je počet sekúnd za minútu. Význam konštanty $1/5$ je vysvetlený vyššie. Konštanta *-1* znamená, že čas sa nepočíta pri stlačení prvej klávesy.

Ďalšou skúmanou charakteristikou je **chybovosť**. Je viacero možných metrík ako chybovosť určiť, vybral som však jednu z tých základnejších, keďže aplikácia nevie porovnať pôvodný a prepísaný text užívateľa a tým sa výber metriky značne zužuje. Chybovosť je teda možné spočítať podľa nasledovného vzorca:

$$EKS\ ER[3] = \frac{EKS}{|T|} * 100\%[1] \tag{2.2}$$

Kde EKS je celkový počet chybových písmen a $|T|$ je dĺžka textu.

Nasledovnou skúmanou charakteristikou je **rovnomernosť, resp. pravidelnosť písania**. Časové rozostupy medzi dvomi stlačenými klávesami sú rozdelené do niekoľkých intervalov. Na každý interval pripadá vo výsledku určitý počet úderov spĺňajúcich medze intervalu (napr. na interval 0-20 milisekúnd pripadá 100 úderov, na interval 20-30 milisekúnd pripadá 50 úderov, atď.). Výstupom tejto charakteristiky je histogram časových rozostupov medzi svomi stlačenými klávesami, ktorý napríklad udáva aj rovnomernosť písania. Je možné z neho odvodiť hneď niekoľko informácií. Pravidelnosť písania sa na histograme prejaví vysokými hodnotami len u niekoľkých intervalov. Z histogramu sa však tak isto dá odvodiť aj rýchlosť písania, ak sú vysoké hodnoty v intervaloch s nízkym časovým rozostupom indikuje to vysokú rýchlosť. Ideálny stav by teda bol nasledovný - vysoké hodnoty v čo najmenšom počte intervalov a zároveň by tieto hodnoty mali byť v histograme čo najviac vľavo, teda v tých intervaloch ktoré vyjadrujú čo najmenšie časové rozostupy medzi dvomi stlačenými klávesami. Na histograme sú viditeľné rozostupy medzi dvomi stlačenými klávesami od 0 až do 2000 ms. Hranice intervalov nie sú zvolené ekvidištantne, a to z toho dôvodu, že nie všetky intervaly sú zastúpené rovnomerne. Napríklad do intervalu s hranicami 200-250 ms

[3] z anglického Erroneous Keystroke Error Rate

zväčša padne viacej kláves ako do intervalu 1000-1500 ms. Preto sú užšie intervaly zvolené práve v nižších časových rozostupoch. Prvý interval má hranice 0 až 40 ms a posledný 1500 až 2000 ms.

Poslednou skúmanou zložkou písania sú **preklepy** užívateľa. Základná myšlienka je rozlíšiť kedy sa jedná o preklep a kedy o preformulovanie slova, keďže medzi preklepy nechceme započítavať aj slovo, ktoré sa užívateľ rozhodol jednoducho zmeniť. Po detekcii preklepu je možné určiť písmená, ktoré v preklepe vystupujú nasledovným pseudokódom:

```
i = pressed_letters.size()-1-2*backspace_counter;
original_letter = pressed_letter[i];
j = pressed_letters.size()-1;
new_letter = pressed_letters.size[j];
```

Pri preklepoch teda určujeme písmeno, ktoré užívateľ napísal chybne (nachádza sa na pozícii i) a písmeno, ktorým užívateľ chybné písmeno opravil (nachádza sa na pozícii j). Vyššie uvedený pseudokód síce vyjadruje základnú myšlienku identifikácia preklepu, no problematika samotných preklepov je ešte o niečo zložitejšia a nachádzajú sa v nej mnoho ďalších obmedzení. Tie sa však zistili až po dotazovaní samotných užívateľov, a preto preklepy budú ešte bližšie diskutované v odpovedajúcej kapitole 5.1.

Kapitola 3

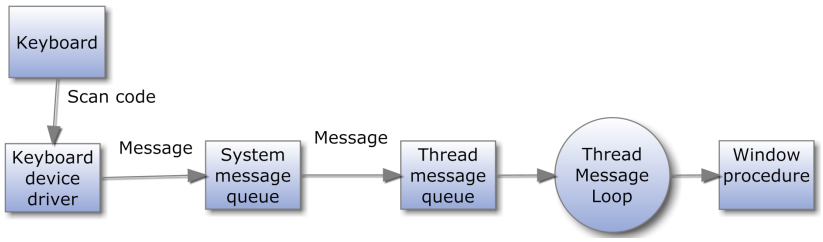
Odchytávanie stlačených kláves na platforme Microsoft Windows

Aplikácia potrebuje klávesy odchytávať v podstate kvôli všetkým skúmaným charakteristikám. Tieto klávesy musí byť schopná odchytávať na globálnej úrovni - t.j. z akéhokoľvek okna, do ktorého užívateľ píše. Vzhľadom na to, že odchytávanie stlačených kláves je platformovo závislá činnosť, budú v tejto kapitole prezentované len možnosti odchytávania na platforme Microsoft Windows 32-bit. Na iných platformách budú tieto postupy pravdepodobne absolútne nepoužiteľné a preto, ak sa niekto bude pokúšať o tvorbu aplikácie podobnej tejto, bude si musieť tieto postupy podrobne naštudovať.

3.1 Model vstupu z klávesnice

Predtým než je možné odchytávať stlačené klávesy je nutné mať poznatky o tom, ako vlastne platforma Microsoft Windows narába so stlačenými klávesami. Tieto poznatky umožňujú pochopiť aj samotné odchytávanie kláves, napríklad pomocou hákov. Takisto pomáhajú pochopiť správanie operačného systému Microsoft Windows pri preklade kláves, ktorý je značne problematický.

Kým aplikácie dostane správu o stlačení klávesy vykoná sa nasledovná postupnosť akcií. Ovládač klávesnice prijme skenovací kód klávesy a preloží ho na virtuálny. Ďalej sa vytvorí správa, ktorá obsahuje skenovací kód, virtuálny kód a ďalšie informácie a táto správa je umiestnená do systémovej fronty správ. Systém z tade tuto správu vyberie a pošle ju vhodnému vláknu. Nakoniec slučka správ daného vlákna odstráni správu a pošle ju do procedúry vhodného okna na spracovanie. Podrobnejší popis tohto modelu sa nachádza na [11] a vystihuje ho nasledovný obrázok.



Obr. 3.1: Grafické znázornenie modelu vstupu z klávesnice

Microsoft Windows zasiela správy o stlačených klávesách implicitne len do okna ktoré je aktívne. Za aktívne okno sa považuje to, na ktoré užívateľ klikol alebo označil pomocou klávesových skratiek ALT+TAB alebo ALT+ESC. Výstupná aplikácia tejto bakalárskej práce však pre svoju činnosť potrebuje informácie o stlačených klávesách na globálnej úrovni. Musí byť teda schopná odchytiť stlačené klávesy aj keď užívateľ túto aplikáciu nemá aktívnu. V nasledujúcich podkapitolách budú vysvetlené možnosti odchytávania kláves na globálnej úrovni. Jednou z nich je použitie okenných hákov, ktorými je možné zachytiť stlačené klávesy ešte predtým, než je informácia o nich doručená aktívnemu oknu. Na vyššie uvedenom obrázku je teda možné si predstaviť akýsi hák, ktorý z "Thread Message Queue" zachytáva správy aj do aplikácie, ktorá má tento hák nainštalovaný.

3.2 GetKeyState, GetAsyncKeyState

Funkcie `GetKeyState` a `GetAsyncKeyState` [9, str. 191-192] slúžia na zistenie stavu klávesy. Vracajú záporné hodnoty, ak je klávesa stlačená. Rozdiel medzi týmito dvomi funkciami je v tom, že funkcia `GetKeyState` odráža stav kláves až do posledného spracovania správ vrátane. Vo väčšine prípadov by to mohlo stačiť, no nie je možné dostávať informácie o klávesách nezávisle na normálnych klávesových správach. Preto je vhodnejšie použiť funkciu `GetAsyncKeyState`, ktorá odráža stav klávesy v skutočne reálnom čase.

Kód na získavanie stlačených kláves v jazyku C by teda mohol vyzerať nasledovne

Algoritmus 3.1: Použitie funkcie GetAsyncKeyState za účelom detekcie a určenia stlačených kláves

```
short character;
while(1) {
  for (character=8;character<=222;character++) {
    if (GetAsyncKeyState(character)==-32767) {
      printf("Bola stlacena klavesa s~ASCII kodom
      %d \n",character);
    }
  }
}
```

Medzi výhody tejto metódy patrí jej jednoduchá implementácia, spoľahlivosť a rýchlosť. Avšak má dve zásadné nevýhody. Tou prvou je, že spotrebúva príliš veľké množstvo procesorového času a výkonu. Na mojom dvojjadrovom AMD Turione X2 RM-70 s frekvenciou 2,00 GHz je to približne 50 %. Táto spotreba by sa dala znížiť zavedením čakania pred while cyklus, no toto riešenie nie je veľmi čisté, vzhľadom na to, že nikde nie je definovaná optimálna dĺžka čakania a mohlo by sa stať, že stlačenia niektorých kláves by ostali nepovšimnuté. Ďalšou nevýhodou je, že podobný kód ako je uvedený vyššie sa častokrát používa v škodlivých programoch typu keylogger [3] a antivírusové programy častokrát hlásia aplikácie obsahujúce takýto alebo podobný úsek kódu ako vírus, čo je samozrejme nežiadúce, keďže moja aplikácia nepredstavuje žiadne bezpečnostné riziko.

3.3 Okenné háky [1]

Mechanizmus hákov je podrobne vysvetlený v [10, str. 866-868]. Pre účely vyvíjanej aplikácie je dôležité vedieť, že tento mechanizmus umožňuje odchytávanie stlačených kláves na globálnej úrovni. Princíp spočíva v nainštalovaní háku na danú udalosť, v našom prípade je touto udalosťou stlačenie klávesnice resp. udalosť vyvolaná myškou. Pri každej takejto udalosti v operačnom systéme sa zavolá nami definovaná obslužná funkcia, ktorá túto udalosť obslúži. Veľká výhoda, oproti odchytávaniu kláves pomocou funkcie `GetAsyncKeyState` je v tom, že sa rapídne zníži zaťaženie systému, pretože program zaťažuje procesor len pri udalosti stlačení klávesy a nie po celý čas svojej činnosti. Implementácia tejto techniky nie je oveľa zložitejšia, jej zjednodušené použitie by v jazyku C (v konzolovej aplikácii) mohlo vyzerať nasledovne:
Ako prvé treba deklarovať **prototyp** obslužnej funkcie:

```
LRESULT CALLBACK keyHandler(int nCode, WPARAM wParam, LPARAM
    lParam);
```

kód funkcie main:

```
HHOOK hookHandle;
//nainštalovanie hooku
hookHandle = SetWindowsHookEx(WH_KEYBOARD_LL, keyHandler, NULL,0);
MSG message;

  while(GetMessage(&message, NULL, 0, 0) != 0) {
    TranslateMessage(&message );
    DispatchMessage(&message );
  }

UnhookWindowsHookEx(hookHandle);
return 0;
```

a samotné **telo obslužnej funkcie** by mohlo vyzerať napríklad takto:

```
KBDLLHOOKSTRUCT *keyPressed = (KBDLLHOOKSTRUCT *)lParam;
  if (nCode == HC_ACTION &&
  (wParam == WM_SYSKEYDOWN || wParam == WM_KEYDOWN))
    printf("Bola stlacena klavesa s ASCII kodom
    %d \n",(int)keyPressed->vkCode);
 return CallNextHookEx(hookHandle, nCode,wParam, lParam);
```

Inštalácia hooku na myšku, by vyzerala obdobne, s tým rozdielom, že identifikátor odchytávaných udalostí by nebol WK_KEYBOARD_LL ale WH_MOUSE_LL. LL v oboch prípadoch znamená Low Level[2].

[1] z anglického Windows Hooks
[2] slovenský preklad - nízka úroveň

3.4 Náčrt problému prekladu kláves

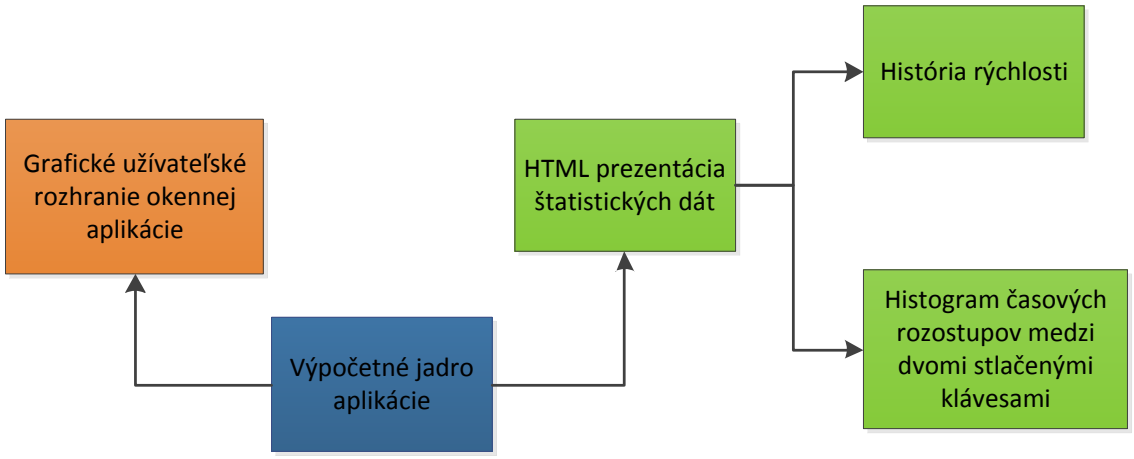
Či už pri technike odchytávania kláves pomocou funkcie GetAsyncKeyState alebo pomocou techniky okenných hákov je hodnota stlačenej klávesy reprezentovaná pomocou virtuálneho kódu klávesy[3] [20]. Tento virtuálny kód korešponduje s anglickým jazykovým rozložením klávesnice. Vzhľadom na to, že aplikácia je vyvíjaná pre užívateľov používajúcich aj slovenské či české jazykové rozloženie, ukážkové zdrojové kódy, spomenuté vyššie by neboli dostačujúce. Po zachytení stlačenej klávesy je teda nutné vykonať preklad tejto klávesy tak, aby jej hodnota odpovedala aktuálnemu jazykovému rozloženiu aktívneho okna. Problematika prekladu kláves je však na platforme windows problematická záležitosť, a to najmä preto, že funkcie, ktoré WinAPI [21] ponúka sú už zastaralé a vyznačujú sa chybovým správaním. Prekladom kláves sa budem podrobne zaoberať v implementačnej časti bakalárskej práce 5.3.1, kde bude podrobne vysvetlený postup aj s ukážkami zdrojových kódov.

[3] z anglického Virtual Key Code

Kapitola 4

Návrh aplikácie

Pri návrhu aplikácie som uplatnil princípy dekompozície, to znamená, že som komplexný problém návrhu aplikácie analyzujúcej užívateľovo písanie rozdelil na niekoľko menších podproblémov. Grafické znázornenie návrhu vyzerá takto:
Obrázok nie je modelovaný pomocou nástrojov UML. Znázorňuje jednotlivé moduly v ap-



Obr. 4.1: Rozloženie problému na podproblémy

likácii a ich náväznosť. Na najnižšej vrstve, neviditeľnej užívateľovi, beží výpočetná časť programu, ktorá sa stará o spracovanie stlačených kláves a ich analýzu. Nad výpočetnou časťou užívateľ vidí grafické užívateľské rozhranie, pomocou ktorého aplikáciu ovláda, a okrem iného, si môže nechať zobraziť webovú prezentáciu štatistických dát, ktorú vygeneruje výpočetná časť programu. Táto prezentácia sa skladá z jednej vstupnej stránky, no obsahuje aj odkazy na podstránky, konkrétne históriu rýchlosti a histogram časových rozostupov medzi dvomi stlačenými klávesami.

4.1 Výpočetné jadro programu

Výpočetné jadro programu je časť programu, ktorá sa stará o programové výpočty. Zachytáva a prekladá stlačené klávesy, analyzuje a ukladá vstupné dáta, resp. metadáta, periodicky vykresľuje ikonku znázorňujúcu aktuálnu užívateľovu rýchlosť písania, generuje

webovú prezentáciu štatistických dát a mnoho ďalšieho. V tejto kapitole sa budem zaoberať návrhom najdôležitejších častí výpočetného jadra.

4.1.1 KeyLogger

Keylogger, teda časť programu zodpovedná za odchytávanie stlačených kláves je jedna z najdôležitejších častí aplikácie, ktorú ku svojej činnosti potrebujú aj ostatné súčasti. Odchytávanie kláves je realizované pomocou okenných hákov, ktorých výhody boli diskutované v 3.3. Okrem háku nainštalovaného na klávesnicu používa aplikácia aj hák nainštalovaný na myšku.

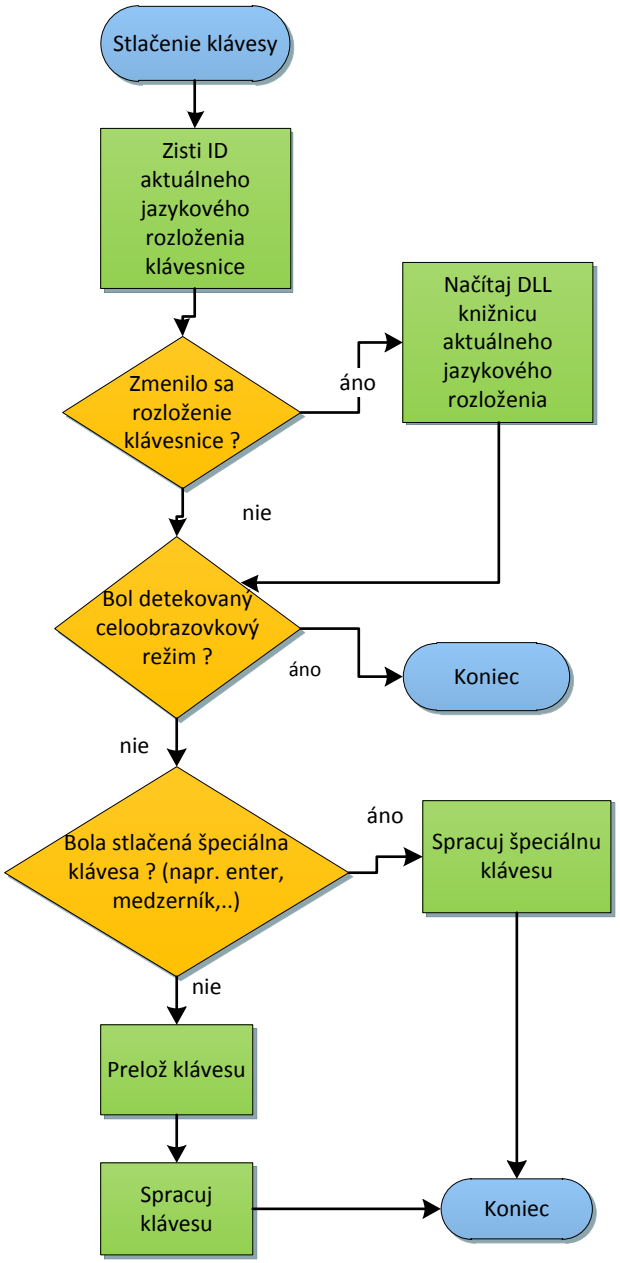
Pri návrhu keyloggeru si bolo nutné uvedomiť niekoľko skutočností, ktoré sa nedali zistiť nijako inak ako empiricky, zo skúseností a pozorovania samotných užívateľov. Napríklad, že nie každú klávesu má zmysel spracovávať, alebo naopak, niektoré klávesy je nutné spracovávať špeciálne. Určite je vhodné ignorovať klávesy typu F1, F2, F3, atď., ako aj PRTSCR, PAUSE, BREAK, INSERT, FN, ALT, START. Naopak, klávesy ENTER, SPACEBAR, TAB majú špeciálny význam, keďže indikujú začiatok nového slova. Ďalšou špeciálnou klávesou je BACKSPACE, jeho výskyt si je nutné niekam poznačiť, keďže táto klávesa môže, no nemusí značiť, že užívateľ urobil preklep. Podobný význam ako klávesa BACKSPACE môže mať aj klávesa LEFT ARROW. Hák na myšku je zavedený z dôvodu, že stlačenie myšky znamená začiatok nového, resp. koniec starého odstavca. Nie je teda potrebné uchovávať si informáciu, ktoré tlačítko na myške bolo stlačené, stačí vedieť, že nastala akcia vyvolaná myškou. Toto všetko sú skutočnosti ktoré je nutné neskôr, pri samotnej implementácii zohľadniť. Je výhodnejšie filtrovať klávesy už pri samotnom zachytení klávesy ako až pri následnej analýze.

Pri návrhu keyloggeru som sa rozhodol zohľadniť aj písanie v celoobrazovkovom režime a to z dôvodu hrania počítačových hier. Keďže nechcem, aby boli štatistiky znehodnotené klávesami, ktoré užívateľ stláčal, keď sa hral, rozhodol som sa takéto klávesy ignorovať. Z podobného dôvodu som sa rozhodol venovať pozornosť aj klávesam W, S, A, D, ktoré sa tiež často používajú pri hraní rôznych počítačových hier. Pri zvýšenej frekvencii týchto kláves, text, ktorý tieto klávesy obsahoval, ignorujem.

V ostatných prípadoch, keď sa jedná o bežnú klávesu, teda písmeno, alebo číslo, nestlačené v celoobrazovkovom režime, si túto klávesu program uloží. Ukladaná je len posledná stlačená klávesa, a to ako z bezpečnostných, tak z pamäťových príčin. Okrem samotnej hodnoty stlačenej klávesy si program pamätá aj časovú značku stlačenia klávesy, ktorá je potrebná pri výpočte histogramu časových rozostupov medzi dvomi stlačenými klávesami, alebo rýchlosti písania. Samozrejme, predtým, než je možné klávesu si uložiť, je nutné vykonať jej preklad. Ako už bolo spomenuté v 3.4, tento preklad je problematická záležitosť, a bude podrobne rozobraná v implementačnej časti 5.3.1 .

Keďže je keylogger realizovaný pomocou mechanizmu hákov, aj pri najrýchlejšom písaní sa mi procesor počítača podarilo zaťažiť len na 1-2 %. Tým pádom je splnený požiadavok na nízke zaťaženie počítača. Program tak isto málo zaťažuje aj pamäť počítača, keďže podrobné informácie si ukladá len o poslednej stlačenej klávese.

Vyššie popísaný návrh je možné vyjadriť nasledovným diagramom.



Obr. 4.2: Vývojový diagram keyloggeru

Proces spracovania klávesy zahŕňa analýzu vstupných dát, ktorá bude rozobraná v nasledujúcej kapitole 4.1.2

4.1.2 Analýza vstupných dát

Ďalšou súčasťou výpočetného jadra je analýza vstupných dát. Tým sa myslí hlavne rozdelenie vstupného textu do úsekov a formát ukladania štatistických dát.

Štatistiky týkajúce sa užívateľovho písania mu sú prezentované až na jeho žiadosť. Preto je nutné počítať s tým, že užívateľ si ich môže pri niekoľko dňovom používaní aplikácie nechať vygenerovať napríklad iba raz. Štatistiky teda musia mať životnosť presahujúcu beh programu. Prvý krok návrhu štatistických dát je však rozdelenie písaného textu do úsekov. Rozhodol som sa pre nasledovné rozdelenie:

- štatisticky nezaujímavý text: 1-2 slová - takýto úsek sa ignoruje
- herný mód - vysoký počet kláves W, S, A, D vo vstupnom texte - takýto úsek sa ignoruje
- text napísaný v celoobrazovkovom režime - takýto úsek sa ignoruje
- krátky paragraf : 3-10 slov - štatisticky zaujímavý úsek
- stredne dlhý paragraf : 11-20 slov - štatisticky zaujímavý úsek
- dlhý paragraf: 20 a viac slov - štatisticky zaujímavý úsek

Pre nasledovné rozdelenie som sa rozhodol z toho dôvodu, že užívateľovo písanie vykazuje pri každom takomto paragrafe mierne odlišné vlastnosti a chcem aby užívateľ mal možnosť si týchto rozdielov všimnúť.
Naopak, úseky, o ktorých som sa rozhodol, že ich budem ignorovať, by štatistiky znehodnocovali, pretože sa nedajú zaradiť medzi normálne písania na klávesnici.

Samotné ukladanie štatistických dát prebieha do textových súborov. Formát txt som zvolil z dôvodu jednoduchej práce s týmito súbormi. Aplikácia si pri spustení načíta dáta obsiahnuté v týchto súboroch do interných štruktúr, ktoré obsahujú presne tie isté položky, ktoré sú v textových súboroch, takže v počítaní štatistík sa pokračuje tam, kde sa prestalo. Štruktúra konkrétneho štatistického súbory vyzerá nasledovne:

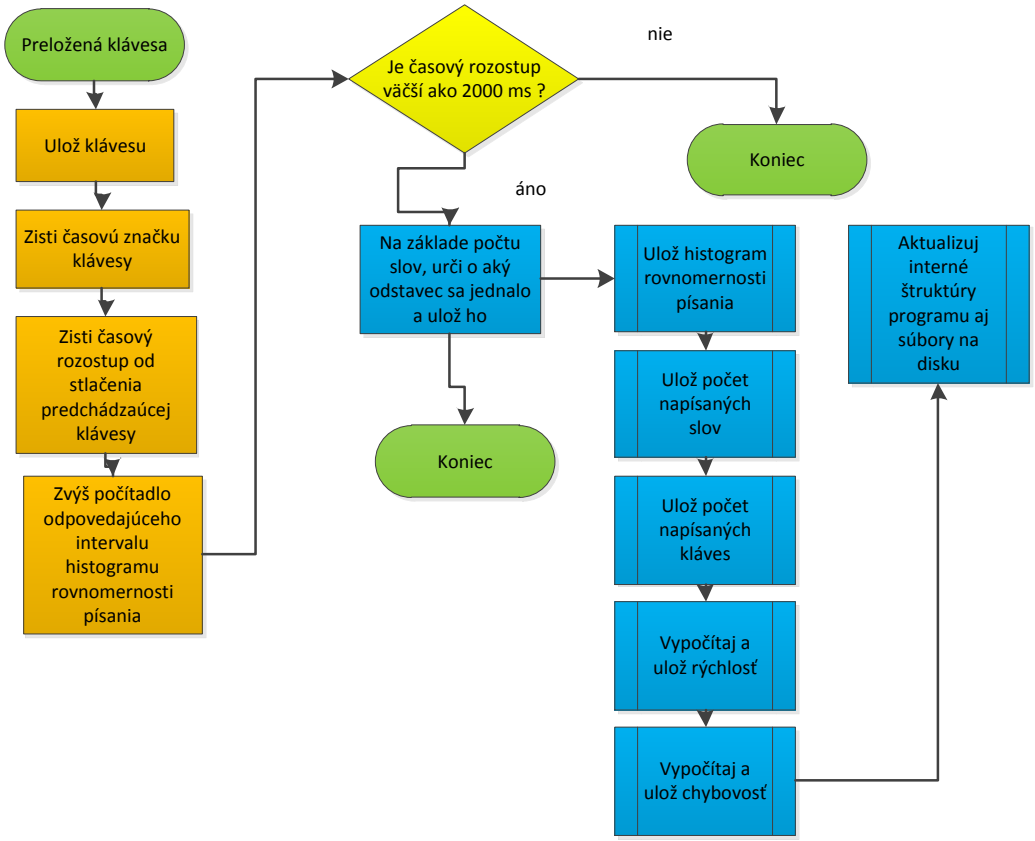
```
# No. of paragraphs: 1121
# No. of words: 5984
# No. of keystrokes: 31432
Average speed: 127019
Average error rate: 93.251
Average word length: 5.25267
Error string: s&rarr;y<br>
1938
3341
4111
4121
4756
4848
1906
1688
945
1183
814
951
535
186
```

```
186
```

Rýchlosť aj chybovosť sa ukladajú kumulatívne, t.j. pri výslednej interpretácii týchto dát je nutné ich hodnoty ešte podeliť počtom paragrafov aby sme dostali skutočné hodnoty. Formát ukladania preklepov je zvolený tak, aby sa daný reťazec preklepu už rovno mohol vypisovať do html súboru. Reťazec `s&rarr;y<br>` uložený ako html bude vo výsledku vyzerať ako `s->y\n`. Každý z pätnástich riadkov, ktoré obsahujú celé čísla, reprezentuje počet výskytov stlačených kláves v konkrétnom intervale histogramu časových rozostupov medzi dvomi stlačenými klávesami.

Na tomto mieste je vhodné zmieniť bezpečnosť aplikácie. Z ukladaných dát je jasné, že na disk sa ukladajú len metadáta. Nie je teda možné, aby mal útočník prístup k citlivým údajom užívateľa ako sú jeho prihlasovacie mená a heslá.

Vstupom spracovania stlačenej klávesy je klávesa preložená modulom KeyLogger. Samotnú činnosť spracovania vyjadruje nasledovný diagram:



Obr. 4.3: Vývojový diagram procesu spracovania stlačenej klávesy

V diagrame je vidieť hodnotu 2000 ms, ktorá reprezentuje hranicu medzi koncom jedného a začiatkom druhého úseku textu. Rozhodol som sa tak na základe empirických meraní, kde som zistil, že aj keď je užívateľ začiatočník v písaní na klávesnici, rozostupy medzi dvomi stlačenými klávesami nepresahujú hodnotu 2000 ms.

4.2 Prezentácia štatistických dát

V predchádzajúcej kapitole bol vysvetlený spôsob a formát ukladania štatistických dát. Užívateľ ale nebude vidieť tieto súbory. Z textových súborov by toho veľa nevyčítal. Preto som sa rozhodol, že štatistické dáta budem prezentovať v jazyku HTML. V tejto forme budú výsledná dáta aj na oko krajšie vyzerať, zvýši sa prehľadnosť a užívateľská prívetivosť. Výsledná webová prezentácia samozrejme využíva textové súbory so štatistikami, no vhodným spôsobom ich formátuje.

Na obrázku 4.1 je prezentácia štatistických dát znázornená zelenou farbou. Pri návrhu webovej prezentácie som sa snažil o prehľadnosť a jednoduchosť. Z tohto dôvodu som sa snažil poskytnúť užívateľovi čo najviac informácií pokope, formou tabuľky, aby sa v nich vedel jednoducho zorientovať. Preto sa na vstupnej stránke nachádzajú základné údaje o jednotlivých druhoch paragrafov a nápoveda. Užívateľ sa preto nemusí nijako bližšie zoznamovať s používaním aplikácie ani so samotným prehliadaním a interpretovaním webovej prezentácie, keďže všetky podstatné informácie má na jednej stránke, a význam týchto informácií má vysvetlený v nápovede. Z priestorových dôvodov nie sú grafy a histogramy umiestnené na hlavnej stránke. Tam sa nachádzajú len odkazy na podstránky, ktoré tieto grafy obsahujú. Vo webovej prezentácii sa zobrazujú užívateľove preklepy. Pri návrhu je zohľadnené, že týchto preklepov môže byť neobmedzené množstvo, preto ich nezobrazujem po načítani hlavnej stránky všetky. Užívateľ vidí len prvé tri najčastejšie preklepy, samozrejme však má možnosť si rozbaliť zoznam preklepov jednoduchým kliknutím. Približný náčrt grafickej podoby prezentácie štatistických dát vyzerá takto:

Smart Typer 2000

Nápoveda

Štatistický údaj	Krátky odstavec	Stredný odstavec	Dlhý odstavec	Súhrn
Počet odstavcov				
Počet slov				
Počet úderov do klávesnice				
Dĺžka slova				
Rýchlosť				
Chybovosť				
Histogram rovnomernosti	Odkaz	Odkaz	Odkaz	Odkaz
História rýchlosti				Odkaz
Preklepy	Rozbaľovací zoznam	Rozbaľovací zoznam	Rozbaľovací zoznam	Rozbaľovací zoznam

Obr. 4.4: Grafický náčrt prezentácie štatistických dát

Z obrázku to nie je zrejme úplne jasné, no nápoveda je implicitne taktiež zabalená

(skrytá), a je viditeľný len jej nadpis. Je to z toho dôvodu, že dĺžka nápovedy je relatívne veľká a jej implicitné rozbalenie by mohlo spôsobiť dezorientáciu užívateľa na stránke. Navyše informácie, ktoré sa nachádzajú na stránke s dátami sú ľahko pochopiteľné aj bez nápovedy. Ak sa však užívateľ rozhodne ju zobraziť, je napísaná tak, aby v nej čo najrýchlejšie našiel nejaké oporné body - t.j. na každom riadku je vysvetlený nejaký pojem, a kľúčového slovo je na danom riadku zvýraznené. Užívateľ tak môže rýchlejšie nájsť to, čo hľadal.

Pri návrhu som zohľadnil aj to, že nie každý užívateľ môže vedieť z čísla, ktoré reprezentuje jeho rýchlosť hneď vycítiť, či je jeho rýchlosť písania vysoká alebo nízka. Preto som zaviedol aj slovné ohodnotenie rýchlosti. Stačí, ak užívateľ prejde myškou ponad súhrnný údaj o rýchlosti, a zobrazí sa mu jeho slovné ohodnotenie. To je napísané tak, aby aj pri nízkej rýchlosti užívateľa motivovalo k zlepšovaniu a neodradzovalo ho to od ďalšieho používania aplikácie.

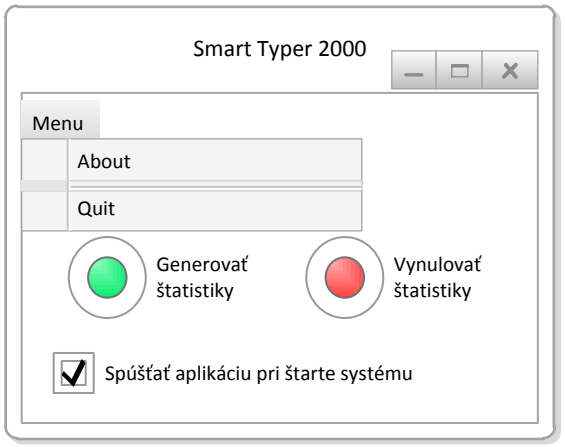
Cieľom grafického návrhu bolo, aby dokázal užívateľ používať aplikáciu, a vyznať sa v prezentácií štatistických dát aj bez dodatočných školení, vysvetľovania alebo čítania súboru Readme. Aplikácia je zameraná na veľkú skupinu užívateľov a preto bolo nutné počítať s tým, že ju budú používať aj počítačovo menej gramotní ľudia a túto skutočnosť pri návrhu zohľadniť.

4.3 Grafické užívateľké rozhranie okennej aplikácie

Síce grafické užívateľské rozhranie okennej aplikácie je to prvé, čo užívateľ vidí keď sa zoznámi s aplikáciou, pri jej ďalšom používaní ho veľmi používať nebude. Je to z toho dôvodu, že aplikácia beží na pozadí, kde monitoruje užívateľove aktivity na klávesnici a preto je zbytočné "investovať" do príliš výrazného grafického rozhrania, keďže užívateľ ho bude používať len na generovanie štatistík. Aplikácia ku svojej činnosti nepotrebuje takmer žiadnu interakciu s užívateľom. Akcií, ktoré užívateľ môže vykonať pri narábaní s aplikáciou skutočne nie je mnoho. Jedná sa o tieto možnosti:

- Generovanie štatistík
- Vynulovanie štatistík
- Spúšťať aplikáciu pri štarte systému
- Minimalizovanie aplikácie
- Ukončenie aplikácie

Užívateľ by preto nemal v ovládaní tápať, cieľom je aby bolo ovládanie natoľko intuitívne, že k nemu užívateľ nebude potrebovať už žiadne ďalšie vysvetlenie. Tieto myšlienky som zhrnul do nasledovného grafického návrhu:



Obr. 4.5: Náčrt grafického užívateľského rozhrania okennej aplikácie

Návrhu dominujú dve veľké ovládacie tlačítka. Ak chce užívateľ zvoliť možnosť spúšťania aplikácie po štarte systému, môže tak urobiť jednoduchým zaškrtnutím políčka. Návrh teda neobsahuje nič navyše, a užívateľovi je jasné, ako sa aplikácia ovláda už pri jej prvom spustení.

Kapitola 5

Implementácia aplikácie

Táto kapitola sa zaoberá technológiami a postupmi, ktoré boli použité pri implementácii. Podrobnejšie budú vysvetlené niektoré algoritmy, aj s ukážkami kódu. Rozhodol som sa, že nebudem popisovať všetky postupy použité pri implementácii programu, ale len tie najzaujímavejšie. Pozornosť bude venovaná aj dotazníkom a testovaniu, keďže sa významnou mierou podieľali na finálnej podobe aplikácii.

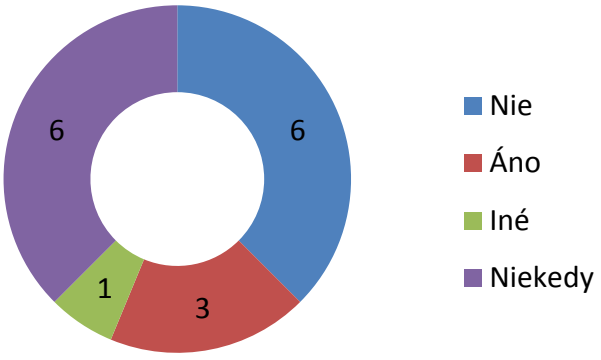
5.1 Dotazníky a testovanie

Aplikácia Smart Typer 2000 je užívateľky zameraná, preto dotazníky a testovanie boli dôležitou súčasťou jej vývoja. Po naprogramovaní prototypu aplikácie, program neustále podliehal testovaniu. Bolo nutné otestovať korektnosť odchytávania kláves na rôznych verziách operačného systému Microsoft Windows ale aj na rôznych typoch klávesníc. Ďalej bolo nutné spracovať pripomienky od užívateľov, aby sa aplikácia čo najviac priblížila ich požiadavkám.

Vytvoril som dotazník s niekoľkými jednoduchými otázkami. Odpovede na tieto otázkami mi pomohli v implementácii niektorých funkcií aplikácie.

Prvá otázka, ktorú som sa užívateľov pýtal bola či zvyknú písať na klávesnici aj v celoobrazovkovom režime.

Po bližšom dotazovaní užívateľov bolo zistené, že odpovede **Áno** a **Niekedy** zväčša zna-



Obr. 5.1: Odpovede na otázku týkajúcu sa písania v celoobrazovkovom režime

menali hranie počítačových hier. Len veľmi málo užívateľov používalo klávesnicu v celooobrazovkovom režime na klasické písanie. Na základe odpovedí na túto otázku som mohol do

aplikácie pridať ignorovanie textu pri detekcii herného módu a detekcii celoobrazovkového režimu.

Ďalšia otázka sa týkala preklepov, konkrétne ako reagujú užívatelia keď zistia, že spravili preklep.

Odpoveď	Počet odpovedí
Backspacom zmažem všetky písmenká pred tým nesprávným+to nesprávne, a pokračujem v písani textu ďalej. napríklad - “Atoj” -> “A” -> “Ahoj:	9
Backspacom zmažem komplet celé slovo a prepíšem ho, napr. “Atoj” -> “” -> “Ahoj”	3
myškou kliknem za nesprávne písmeno a vymažem ho pomocou tlačítka BACKSPACE, napr. “Atoj” -> ”Aoj” -> “Ahoj”	1
Myškou kliknem pred nesprávne písmeno, a vymažem ho pomocou DELETE	0
V slove v ktorom je preklep, sa posúvam pomocou šípky, a písmeno vymažem tlačítkom BACKSPACE	0
V slove v ktorom je preklep, sa posúvam pomocou šípky, a písmeno vymažem tlačítkom DELETE	0
Iné	3

Tabuľka 5.1: Vyhodnotenie otázky ’Ako zareagujete, keď zistíte, že ste spravili preklep?’

Z odpovedí je jasne vidieť, že väčšina užívateľov používa pri oprave chyby klávesu BACKSPACE, preto ani nie je potrebné brať klávesu DELETE pri preklepoch do úvahy.

Ďalšia otázka sa pýtala na to, ako užívatelia reagujú, ak chcú slovo preformulovať. Už v kapitole, ktorá popisuje keylogger je zmienka o tom, že je nutné rozlíšiť preklep a preformulovanie slova.

Odpoveď	Počet odpovedí
Backspacom mažem slovo písmenko po písmenku	11
označím slovo myškou, a zmažem ho pomocou jedného stlačenia BACKSPACE	11
označím slovo myškou a zmažem ho pomocou jedného stlačenia DELETE	2
Myškou kliknem pred nesprávne písmeno, a vymažem ho pomocou DELETE	0
Iné	3

Tabuľka 5.2: Vyhodnotenie otázky ’ Čo urobíte, keď chcete preformulovať slovo, ktoré ste práve napísali ?’

Z odpovedí je vidieť, že klávesa BACKSPACE sa veľmi často používa aj pri preformulovaní slova. Je preto nutné tieto dve situácie nejako rozlíšiť. Ak je počet BACKSPACOV

väčší alebo rovný ako dĺžka aktuálne uložených stlačených kláves, nejedná sa o preklep alebo o preformulovanie. Pre väčšiu názornosť uvediem príklad.

- Užívateľ napíše 'Atoj', postupným mazaním mu ostane 'A', a slovo dopíše na 'Ahoj' - bol detekovaný preklep
- Užívateľ napíše 'Ahoj', postupným mazaním mu ostane (prázdny reťazec) a slovo dopíše ako 'Dobrý deň' - bolo detekované preformulovanie slova

Ďalšie názory som už od užívateľov nezbieral formou dotazníku. Aplikáciu som posielal užívateľom na otestovanie a zbieral ich názory, pripomienky, pochvaly.
Užívatelia ocenili prehľadnosť webovej prezentácie, jednoduchosť desktopovej aplikácie, a grafické znázornenie niektorých charakteristík formou grafov. Ako nedostatok vzišla málo viditeľná nápoveda na stránke, aj preto je vo finálnej verzii grafického návrhu webovej prezentácie zvýraznená. Medzi ďalšie návrhy patrilo doplnenie počtu výskytov daného preklepu alebo pridanie slovného ohodnotenia ku rýchlosti. Okrem návrhov bolo testovanie aplikácie užívateľmi prospešné aj v objavovaní chýb, ktoré som si pri vyvíjaní na vlastnej platforme nevšimol, a niekedy ani nemohol všimnúť. Sem patrí napríklad nezobrazovanie obrázkov v aplikácii, vďaka chýbajúcim knižniciam na strane užívateľa. Vďaka testovaniu som tento problém mohol včas objaviť, a požadované knižnice spolu s aplikáciou pribaliť.

5.2 Použité technológie, nástroje, knižnice

Pri vývoji aplikácii bolo použité veľké množstvo rôznych technológií. Základným vývojovým prostredím bolo Qt SDK[1] [2] vo verzii 1.2. Obsahuje Qt Creator, Qt Libraries, Qt Development Tools, a ďalšie nástroje potrebné k vývoji aplikácie. Samotné Qt je medziplatformový aplikačný framework, ktorý sa používa hlavne pri vývoji aplikácií s grafickým užívateľským rozhraním. Medzi jeho výhody určite patrí rýchly vývoj grafického užívateľského rozhrania, bohaté knižnice, či kvalitná dokumentácia. Qt je postavené nad jazykom C++, okrem klasických konštrukcií jazyka C++ a používania tried frameworku Qt sú v zdrojových kódoch aj volania funkcií Windows API[21].

Pri návrhu prezentácie štatistických dát boli použité technológia HTML, CSS, JavaSript. Grafy a histogramy sú vytvorené pomocou Google Visualization API [13]. Webové stránky so štatistikami sú štýlované pomocou Twitter Bootstrap [3] .

5.3 Vybrané algoritmy a postupy

Táto podkapitola sa bude zaoberať implementačnou stránkou vývoja aplikácie, nebudú tu spomenuté všetky postupy, diskutované budú len tie najzaujímavejšie, poprípade netriviálne.

5.3.1 Preklad kláves na platforme Microsoft Windows

Návratová hodnota pri odchytávaní stlačenej klávesy, či už pri použití funkcie `GetAsyncKeyState` alebo pomocou techniky *Windows Hooks*, je *Virtual Key Code*. Napríklad pre klávesu *4*, na anglickej klávesnici je to `0x34`, problém je, že na českej alebo

[1] http://qt.nokia.com/
[2] SDK - Software Development Kit
[3] http://twitter.github.com/bootstrap/

slovenskej klávesnici je tento istý kód vrátený aj pri stlačení písmena *'č'*. Rozlíšenie správnej klávesy aplikácia potrebuje pri správnom identifikovaní preklepov. Ako vidieť, Virtual Key Codes korešpondujú s anglickým klávesovým rozložením. Čo teda robiť, ak má užívateľ nastavené iné rozloženie? Stlačenú klávesu je nutné, na základe aktuálneho jazykového rozloženia, a stavu niektorých iných kláves preložiť na odpovedajúcu klávesu.

Operačný systém Microsoft Windows ponúka niekoľko funkcií, ktoré tento preklad vykonajú. Konkrétne sú to funkcie `ToAsciiEx`, `ToUnicodeEx`. Tieto funkcie pracujú korektne s väčšinou kláves, problém nastáva pri práci s mŕtvymi klávesami, to sú tie, ktoré samy o sebe neprodukujú žiadny výstup, používajú sa však pri tvorbe znakov s diakritikou. Na slovenskej klávesnici je to najčastejšie mäkčeň - '˘', a dĺžeň - '´'. Tieto funkcie nie sú schopné zložiť výsledný znak pri stlačení mŕtvej klávesy. Na stránkach dokumentácie týchto funkcií je napísané

> 'The parameters supplied to the **ToAsciiEx** function might not be sufficient to translate the virtual-key code, because a previous dead key is stored in the keyboard layout.'[19].

Jediný problém však nie je nekorektný preklad. Táto funkcia tak isto absolútne znehodnocuje užívateľovo písania na klávesnici nasledovným spôsobom. Užívateľ stlačí mŕtvu klávesu, napríklad '´', prekladová funkcia `ToAsciiEx`, pri stlačení ďalšej klávesy, na ktorú chcel užívateľ aplikovať mŕtvu klávesu, napríklad 'a', aplikuje mŕtvu klávesu sama na seba a nie na klávesu 'a'. Výstup, ktorý užívateľ vidí, ak má zapnutú aplikáciu, ktorá prekladá klávesy pomocou funkcie `ToAsciiEx` je '´´a' namiesto 'á'. Keďže dokumentácia tejto, funkcie sa o tejto chybe ďalej nezmieňuje, je neprijateľné používať funkcie `ToAsciiEx` alebo `ToUnicodeEx` v nezmenenej podobe.

Prvou myšlienkou bolo si pri stlačení mŕtvej klávesy uložiť ukazateľ na štruktúru KBDLLHOOKSTRUCT [14], ktorá obsahuje informácie o stlačených klávesách, teda aj virtuálnych kód stlačenej klávesy. Potom počkať, až dokým funkcia `ToAsciiEx` vráti nejakú inú hodnotu ako -1 a následne vložiť uložený virtuálny kód naspäť a vykonať preklad. Síce sa podarilo odstrániť zasahovanie do výstupu užívateľa, informácia o preloženej klávese ale nebola úplná, t.j. namiesto 'á' sa uložilo 'a'.

Ďalším riešením bolo urobiť si tento preklad manuálne. To znamenalo zisťovanie stavu kláves SHIFT a CAPS LOCK, ďalej zisťovanie jazykového rozloženia klávesnice, následné overovanie, či aktuálne stlačená klávesa nemá virtuálny kód, ktorý by potencionálne mohol reprezentovať mŕtvu klávesu a potom samotný preklad. Toto riešenie fungovalo pre väčšinu kláves, ale len pre to jazykové rozloženie, ktorého podpora bola naprogramovaná. Navyše virtuálny kód mŕtvych kláves sa mohol líšiť aj medzi českým a slovenským rozložením a dokonca aj medzi dvomi rôznymi druhmi klávesníc, pretože štandard nestanovuje vysoké hodnoty virtuálny kódov, ponecháva ich priradenie daným klávesám na výrobcoch klávesníc. Preto som toto riešenie označil za nesystematické a navyše veľmi zdĺhavé.

Posledné riešenie, ktoré je použité aj vo výslednej aplikácii už dáva uspokojivé výsledky na väčšine známych jazykových rozloženiach. Nebolo testované na rozloženiach, ktoré používajú inú abecedu, úpravou kódu by sa však malo dať dosiahnuť aj podpory týchto rozložení. Výsledné riešenie je založené na tomto článku [5]. Riešenie prezentované v článku je náhradou funkcie `ToUnicode`, aplikácia Smart Typer 2000 však ku svojej činnosti potrebuje riešenie, ktoré by nahradzovalo správanie funkcie `ToUnicodeEx`. Je nutné počítať s tým, že užívateľ môže jazykové rozloženia klávesnice meniť, a na túto zmenu je schopná reagovať práve funkcia `ToUnicodeEx`. Preto bol kód v spomínanom článku upravovaný pre potreby aplikácie.

Prvým krokom je teda zistenie aktuálneho jazykového rozloženia klávesnice, ako sa toho docieli je možné pochopiť z nasledujúceho úseku kódu:

Algoritmus 5.1: Získanie aktuálneho jazykového rozloženia klávesnice

```
// keyboard name
 char kbdName[KL_NAMELENGTH];
 // active input locale identifier
 HKL kb_input;
// Structure containing information about a GUI thread
 GUITHREADINFO info;
 info.cbSize = sizeof( GUITHREADINFO );
 DWORD thread_id;

 // get information about current thread
 GetGUIThreadInfo(0,&info);
  // handle of window which has active keyboard focus
 thread_id = GetWindowThreadProcessId(info.hwndFocus,0);
 // get input language ID of this window
 kb_input = GetKeyboardLayout(thread_id);
 // set layout we obtained to our application
 ActivateKeyboardLayout(kb_input,0);
 //  Retrieves the name of the active input locale identifier
 GetKeyboardLayoutNameA(kbdName);
```

V premennej `kbdName` je teda uložený názov aktuálneho jazykového rozloženie. Pod pojmom názov sa však nerozumie slovný názov, ako napr. 'US_KB', je to číselná konštanta, ktorá jednoznačne identifikuje daný jazyk [15].

Ďalším krokom je zistenie názvu DLL knižnice v ktorej sú uložené informácie o aktuálnom jazykovom rozložení a jeho klávesách. Tento názov sa nachádza v registroch, na adrese `SYSTEM\\Control\\Keyboard Layouts\\`**`kbdName`** [4, str. 118]. Samotný názov potom obsahuje položka `Layout File`. Zápis v jazyku C by vyzeral nasledovne

Algoritmus 5.2: Získanie názvu DLL knižnice, ktorá obsahuje informácie o aktuálnom jazykovom rozložení

```
 HKEY hKey;
 char kbdKeyPath[51 + KL_NAMELENGTH];
 snprintf(kbdKeyPath, 51 + KL_NAMELENGTH,
"SYSTEM\\CurrentControlSet\\Control\\Keyboard Layouts\\
 %s",kbdName);
 if(RegOpenKeyExA(HKEY_LOCAL_MACHINE, kbdKeyPath, 0,
    KEY_QUERY_VALUE, &hKey) != ERROR_SUCCESS) {
 return -1;
}
 if(RegQueryValueExA(hKey, "Layout File", NULL, &varType,
    (BYTE*)layoutFile,&bufferSize) != ERROR_SUCCESS) {
    return -1;
}
```

```
RegCloseKey(hKey);
return 1;
```

`layoutFile` je parameter funkcie `int getKeyboardLayoutFile(char* layoutFile, DWORD bufferSize)`.

Pri slovenskom jazykovom rozložení je hodnota `kbdName` 0001041b a v kľúči Layout File je hodnota KBDSL1.DLL.

Ďalej je potrebné získať absolútnu cestu k danej DLL knižnici a následne túto knižnicu načítať. V zdrojových kódoch túto úloho plní funkcia `HINSTANCE loadKeyboardLayout()`.

Najprv načítame názov DLL knižnice pomocou nasledovného volania

```
char layoutFile[MAX_PATH];
if(getKeyboardLayoutFile(layoutFile, sizeof(layoutFile)) == -1)
return NULL;
```

Potom získame absolútnu cestu k danej knižnici

Algoritmus 5.3: Získanie absolútnej cesty danej DLL knižnice

```
HINSTANCE kbdLibrary;
char systemDirectory[MAX_PATH];
GetSystemDirectoryA(systemDirectory, MAX_PATH);
char kbdLayoutFilePath[MAX_PATH];
snprintf(kbdLayoutFilePath, MAX_PATH, "%s \\ %s",
systemDirectory, layoutFile);
```

Danú knižnicu načítame

```
kbdLibrary = LoadLibraryA(kbdLayoutFilePath);
```

Ďalej je potrebné získať adresu funkcie KbdLayerDescriptor, táto funkcia sa nachádza už v spomínanej DLL knižnici. Bohužiaľ, Microsoft k tejto funkcii neponúka žiadnu dokumentáciu.

Algoritmus 5.4: Získanie adresy funkcie KbdLayerDescriptor

```
KbdLayerDescriptor pKbdLayerDescriptor = NULL;
pKbdLayerDescriptor = (KbdLayerDescriptor)GetProcAddress(kbdLibrary
,"KbdLayerDescriptor");
```

Následne je nutné naplniť štruktúry PKBDTABLES volaním funkcie, ktorej adresa bola práve získaná.

```
PKBDTABLES pKbd;
pKbd = pKbdLayerDescriptor();
```

Štruktúra `PKBDTABLES` ale aj ďalšie iné sú prevzaté z Windows Driver Kitu [22], konkrétne z hlavičkového súboru kbd.h.

Samotný preklad stlačenej klávesy potom prebieha nasledovne. Modul keylogger dostane informáciu o stlačenej klávese. Zistí, či sa jazykové rozloženie od poslednej stlačenej klávesy zmenilo, ak áno zavolá sa funkcia `HINSTANCE loadKeyboardLayout()`, ktorá obsahuje aj volanie funkcie `int getKeyboardLayoutFile(char* layoutFile, DWORD bufferSize)`. Preklad stlačenej klávesy potom vykonáva funkcia

`int convertVirtualKeyToWChar(int virtualKey, PWCHAR outputChar, PWCHAR deadChar)`, ktorá postupne prehľadáva tabuľky, ktoré sa naplnili volaním funkcie `pKbdLayerDescriptor()`.
Konkrétne sa jedná o nasledovné makro, ktoré je oproti originál verzii upravené tak, aby korektne pracovalo aj pri prepínaní jazykových rozložení:

Algoritmus 5.5: Hľadanie stlačenej klávesy v tabuľkách

```
#define SEARCH_VK_IN_CONVERSION_TABLE(n)\
i = 0;\
if (pVkToWchars##n && (mod < n)) {\
  do  {\
    if (pVkToWchars##n[i].VirtualKey == virtualKey \
       && charCount == 0) {\
      if (pVkToWchars##n[i].Attributes == CAPLOK && capsLock) {\
        if (mod == shift) mod = 0;\
        else mod = shift; \
      }\
      if (pVkToWchars##n[i].wch[mod]) {\
        * outputChar = pVkToWchars##n[i].wch[mod];\
        charCount = 1;\
      }\
      if ( * outputChar == WCH_NONE) {\
        charCount = 0;\
      }\
      else if ( * outputChar == WCH_DEAD) {\
        if (pVkToWchars##n[i + 1].wch[mod]) {\
          * deadChar = pVkToWchars##n[i + 1].wch[mod];\
          charCount = 0;\
        }\
      }\
      break;\
    }\
    i++;\
  }\
  while (pVkToWchars##n[i].VirtualKey != 0);\
}\
```

Funkcia je schopná prekladať aj klávesy, ktoré vznikli kompozíciou mŕtvej klávesy a následným stlačením regulárnej klávesy. Zdrojové kódy **kbdext.cpp** a **kbdext.h** som kvôli zachovaniu autenticity upravoval čo najmenej. Nutné úpravy boli tie, aby bol tento algoritmus schopný reagovať na zmenu jazykového rozloženia a s tým spojené opravovanie chýb. Výsledok prekladu stlačenej klávesy je uložený v premennej `outputChar`.

5.3.2 Automatické spúšťanie aplikácie po štarte systému

Aby aplikácia dávala štatisticky presné výsledky, potrebuje čo najväčší vstupný objem dát. Toho sa môže docieliť napríklad tým, že aplikácia sa spustí hneď po štarte systému. Tým pádom bude mať aplikácia k dispozícii kompletnú užívateľovu aktivitu na klávesnici

od spustenia systému až po jeho vypnutie. Implicitne je však táto možnosť nepovolená, aby užívateľ nemal pocit, že aplikácia mu zasahuje do operačného systému bez jeho vedomia. Aktivovanie tejto možnosti je však jednoduché, stačí len zaškrtnúť jedno políčko.

Automatické spúšťanie aplikácie je možné aktivovať zásahom do registru systému Windows. Konkrétne sa jedná o register `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows \CurrentVersion\Run` [16]. Do tohto registru je potrebné pridať položku vo formáte "názov – absolútna cesta ku programu, ktorý chceme po štarte spúšťať".

Aby však hocijaká aplikácia mohla modifikovať registre systému je nutné, aby mala oprávnenia správcu systému. Tieto oprávnenia však nemôže mať priamo aplikácia Smart Typer 2000, pretože systém Windows nepovoľuje automatické spúšťanie aplikácií po štarte, ktoré majú oprávnenia správcu systému. Na tieto účely slúži aplikácia **regapp.exe**, ktorá už oprávnenia administrátora má. Pri aplikovaní administrátorských oprávnení na danú aplikáciu je postup nasledovný:

1. Stiahnutie programu mt.exe [17]
2. Vytvorenie manifest súboru [12], ktorý bude obsahovať nasledovný riadok
   `<requestedExecutionLevel level="requireAdministrator" uiAccess="false"/>`
3. Vytvorenie skriptu UAC.bat s nasledujúcim obsahom
   `mt -manifest cesta/k/manifest/suboru.manifest -outputresource:cesta/k/aplikacii.exe;1`
4. Spustenie skriptu UAC.bat. Aplikácia, ktorej cestu sme zvolili má oprávnenia správu systému.

Signál zmeny zaškrtávacieho políčka, ktorým užívateľ nastavuje automatické spúšťanie aplikácie pri štarte systému je spojený so slotom
`void MainWindow::run_on_startup()` Sú dve možnosti zmeny, buď užívateľ políčko zaškrtol, v tom prípade sa vykoná tento kód:

Algoritmus 5.6: Spustenie aplikácie, ktorá pridá záznam do registrov

```
//if state of checkbox was changed, and it is ticked
if (ui->checkBox->checkState()) {
  if (path.length() == 0 || (path.compare(pom) != 0)) {
    arguments << "--add";
    arguments << registryPath;
    //start program that will add entry to the registry
    process_add->start(AppToExec, arguments);
  }
}
```

Ak užívateľ checkbox odšktrol, tak sa vykoná tento kód:

Algoritmus 5.7: Spustenie aplikácie, ktorá odstráni záznam z registrov

```
 else { //if state of checkbox was changed, and it is unticked
```

```
    if (path.length() > 0) { // && there is something to remove
      arguments << "--remove";
      //start program that will remove entry from registry
      process_remove -> start(AppToExec, arguments);
    }
  }
```

Použité premenné sú deklarované nasledovne

Algoritmus 5.8: Deklarácia použitých premenných

```
QObject * parent_add = new QObject();
QObject * parent_remove = new QObject();
//process that will add entry to the registry
QProcess * process_add = new QProcess(parent_add);
//process that will remove entry from the registry
QProcess * process_remove = new QProcess(parent_remove);
QStringList arguments; //list of arguments
// path of the executable
QString AppToExec = qApp -> applicationDirPath() + "/regapp.exe";
```

Premenná registryPath je prvý argument príkazového riadka
`QString registryPath = registryPath = QString::fromLatin1(argv[0]);`
a premenná path je definovaná ako `QString path = get_path();` .
Funkcia `get_path()` slúži na získanie uloženej hodnoty v registroch, overuje, či už o aplikácii nejaký záznam nie je.

Algoritmus 5.9: Funkcia, ktorá hľadá záznam o aplikácii v registroch, a vracia uloženú cestu

```
QString MainWindow::get_path() {
  QSettings settings("HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\
 Windows\\CurrentVersion\\Run", QSettings::NativeFormat);
  QStringList keys;
  QString path = "";
  keys = settings.childKeys(); //load all keys from the path above
  for (int i = 0; i < keys.size(); i++) {
  //look for entry of our application  in registry and parse it
    if (keys[i].contains("Smart Typer 2000")) {
      path = settings.value(keys[i]).toString();
      path = path.mid(1, path.lastIndexOf("\\"));
      path = path.replace("\\", "/");
    }
  }
  return path;
}
```

Aplikácia Smart Typer 2000 pri zmene stavu políčka zavolá program `regapp.exe` s odpovedajúcim parametrom. Aplikácia `Regapp.exe` je tiež napísaná v Qt/C++, ale je zakázané zobrazovanie jej hlavného okna a po zápise do registrov svoju činnosť automaticky

ukončí. Pridanie záznamu do registru je vykonané takto:

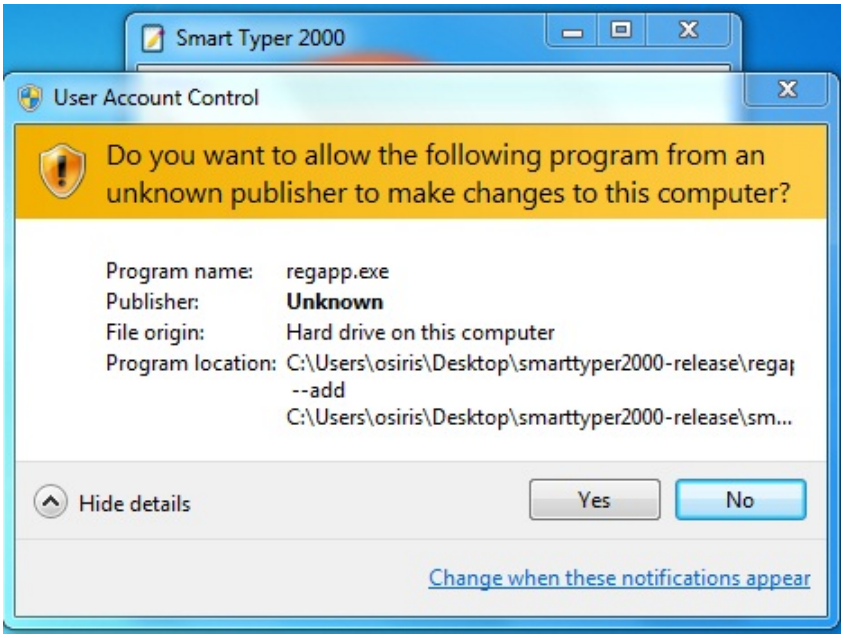
Algoritmus 5.10: Pridanie záznamu do registrov vo formáte "názov,cesta"

```
QSettings settings("HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\
Windows\\CurrentVersion\\Run", QSettings::NativeFormat);
QString apostroph = " \" ";
path.replace("/", "\\");
path = apostroph + path + apostroph + " --argument";
settings.setValue("Smart Typer 2000", path);
```

Premenná `path` je argument príkazového riadka, s ktorým bola aplikácia regapp.exe zavolaná
`path = QString::fromLatin1(argv[2]);`
Odstránenie záznamu z registrov prebieha obdobne:
`settings.remove("Smart Typer 2000");`

Operačný systém Microsoft Windows pri štarte systému prehľadá všetky záznamy, ktoré sú uložené v kľúči Run a spustí všetky aplikácie, ktorých cesta je v zázname uložená.

Na nasledujúcom obrázku je vidieť ako prebieha interakcia programu a užívateľa, keď sa rozhodne zvoliť možnosť automatického spúšťania aplikácie.



Obr. 5.2: Chovanie programu pri zvolení možnosti 'Spúšťať aplikáciu pri štarte systému'

5.3.3 História rýchlosti

Aby mal užívateľ možnosť sledovať vývoj rýchlosti písania v čase, rozhodol som sa implementovať funkciu histórie rýchlosti. Pri implementácii však bolo nutné vysporiadať sa s niekoľkými problémami. Ako prvé bolo nutné zabezpečiť ukladanie záznamov o rýchlosti

na disk. Rozhodol som sa, že užívateľ bude mať možnosť prezrieť si len posledných 31 záznamov a to z dôvodu prehľadnosti výsledného grafu. Ďalším problémom bolo, čo ak užívateľ bude používať aplikáciu v daný deň len krátky čas, a zaznamenaná rýchlosť bude vypočítaná len z jedného alebo dvoch úsekov textu, v takomto prípade by takýto údaj nebol veľmi vierohodný, a mohol by skresľovať reálnu rýchlosť písania užívateľa. Výsledná implementácia je preto nasledovná:

1. Periodické kontrolovanie napísaného textu. Na tieto účely výborne poslúžila trieda `QTimer`[18]

Algoritmus 5.11: Periodická aktualizácia rýchlosti

```
QTimer *timer = new QTimer(this);
connect(timer , SIGNAL(timeout()), this ,SLOT(update_speed()));
timer->start(1800000);
```

Interval kontroly som nastavil na 30 minút. Rozhodol som sa tak preto, aby nedochádzalo ku príliš častým kontrolám, keď by možno nedošlo k výraznej zmene rýchlosti, no na druhej strane sa nestane, že by aplikácia nezaznamenala dlhší úsek písania.

2. Pri každom aktualizovaní rýchlosti najprv overím či už je napísané dostatočné množstvo textu, inak sa v aktualizácii nepokračuje.

Algoritmus 5.12: Kontrola objemu napísaného textu

```
if (paragraph_counter[0] >= 8) {// #No. of short paragraphs
  if (paragraph_counter[1] >= 4) {// #No. of medium paragraphs
    if (paragraph_counter[2] >= 2) {// #No. of long paragraphs
      speed_history_output =
      (speed_history[0] / paragraph_counter[0] +
       speed_history[1] / paragraph_counter[1] +
       speed_history[2] / paragraph_counter[2]) / 3;
      update_flag = true;
    }
  }
}
```

Počty paragrafov, v poradí krátky, stredný, dlhý som nastavil na 8,4, resp. 2. Tieto čísla udávajú minimálny počet paragrafov kedy sa vykoná aktualizácia rýchlosti a zabraňujú skresleniu štatistiky.

3. Zistenie, či sa v daný deň už uložil záznam o rýchlosti na disk.

Algoritmus 5.13: Zistenie, či sme v daný deň, už záznam o rýchlosti uložili

```
QStringList lines;
//get current date
```

```
QDate date = QDate::currentDate();
//convert it to the string
QString dateString = date.toString("dd.MM.");
if (speed_file.open(QIODevice::ReadWrite)) {
  if (speed_file.size() != 0) {
    while (!out.atEnd()) {
    //read whole file - line by line
      lines.append(out.readLine());
    }
    out.flush();
    speed_file.close();
    //save the last line
    lastLine = lines[lines.size() - 1];
    if (dateString.compare(lastLine
    .mid(lastLine.indexOf("->") + 2,-1)) == 0) {
      //re-write the original speed record
    }
```

Vyššie uvedené porovnanie počíta s uloženými záznamami o rýchlosti vo formáte `rýchlost'->DD.MM.`

4. Ak už v daný deň bol záznam o rýchlosti vykonaný, tak sa tento záznam aktualizuje. Posledný záznam sa nachádza na poslednom riadku súboru.

Algoritmus 5.14: Aktualizovanie záznamu o rýchlosti

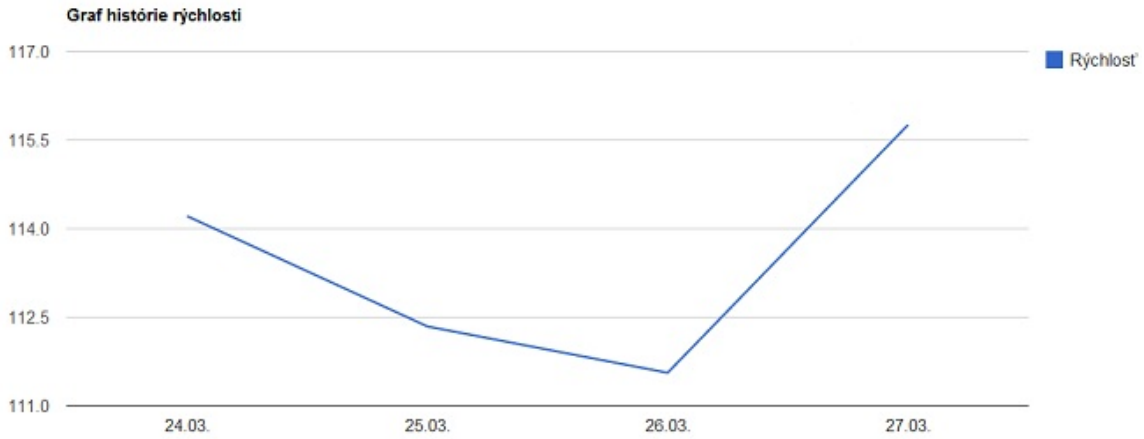
```
lines[lines.size()-1] = QString::number(speed_history_output);
lines[lines.size()-1] += "->";
lines[lines.size()-1] += lastLine.mid(lastLine
                         .indexOf("->") + 2, -1);
```

5. Ak ešte v daný deň nebol uložený záznam o rýchlosti tak ho dopíšeme na koniec súboru

Algoritmus 5.15: Pridanie nového záznamu o rýchlosti (ak sme ešte v daný deň nepridali záznam o rýchlosti)

```
speed_file.open(QIODevice::ReadWrite |
                QIODevice::Append    |
                QIODevice::Text);
line = "\n";
line += QString::number(speed_history_output);
line += "->";
line += dateString;
out << line;
```

Kroky 2 až 5 sa opakujú 30 minút počas celej doby behu programu. Čím dlhšie bude program v daný deň bežať, tým presnejší bude údaj o rýchlosti v daný deň. Výsledný vygenerovaný graf s použitím Google Visualizationi Api vyzerá takto:



Obr. 5.3: Graf histórie rýchlosti obsahujúci záznamy za posledné 4 dni.

5.4 Vizuálna stránka programu

Do vizuálnej stránky programu patrí dizajn grafického užívateľského rozhrania okennej aplikácie a takisto aj vizuálna podoba HTML stránky s prezentáciou štatistík. Na obrázku 4.1 sú to časti vyznačené oranžovou a zelenou farbou.

Grafická podoba HTML prezentácie vychádza z náčrtu **??**. Pri implementácii prezentácie som sa rozhodol použiť technológiu Twitter Bootstrap, čo je v podstate CSS súbor, ktorý aj v predvolených nastaveniach stránku štýluje do užívateľsky prijateľnejšej podoby. Rozdiel je viditeľný na nasledovnom obrázku:

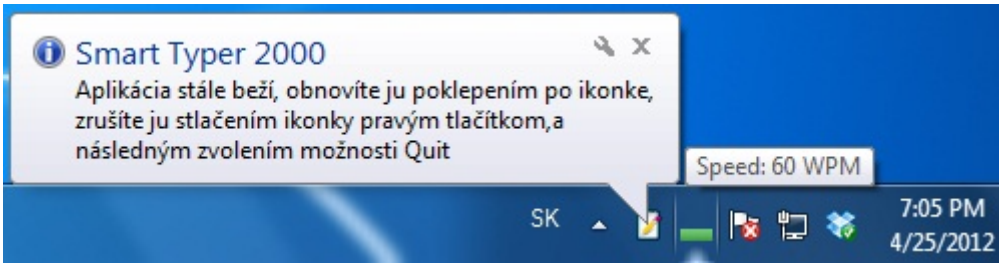
Štatistický údaj	Krátke odstavce
Počet odstavcov	1623
Počet slov	8748
Počet úderov do klávesnice	46911
Dĺžka slova	5.36
Rýchlosť	113.73
Chybovosť	0.08

Štatistický údaj	Krátke odstavce
Počet odstavcov	1623
Počet slov	8748
Počet úderov do klávesnice	46911
Dĺžka slova	5.36
Rýchlosť	113.73
Chybovosť	0.08

Histogram rovnomernosti písania Klikni pre zobrazenie

Obr. 5.4: Porovnanie webovej prezentácie s použitím Twitter Bootstrap (vľavo) a bez (vpravo).

Grafický dizajn okennej aplikácie sa v podstate nelíši od návrhu 4.3. Na obrázku však nie je viditeľný beh aplikácie na pozadí ani ikonka, ktorá v reálnom čase znázorňuje rýchlosť. Táto ikonka sa každé tri sekundy aktualizuje a podľa aktuálnej rýchlosti sa mení intenzita a objem jej výplne. Užívateľ teda môže vidieť svoju aktuálnu rýchlosť a keď cez ikonku prejde myškou, ukáže sa mu aj číselná hodnota rýchlosti. Ikonka má nasledovnú grafickú podobu:



Obr. 5.5: Beh aplikácie na pozadí a ikonka znázorňujúce aktuálnu rýchlosť

Kapitola 6

Záver

Aplikácia Smart Typer 2000 analyzuje užívateľove písanie na klávesnici. Na základe pozitívnych ohlasov od ľudí, ktorí si aplikáciu testovali si dovolím tvrdiť, že sa to podarilo. Samozrejme, aplikácia nie je určená pre každého, no ľudí, ktorých zaujíma ich písanie na klávesnici a štatistiky s ním spojené nájdu aplikáciu užitočnú. Aplikácia zbiera dáta o písaní užívateľa na klávesnici a následne ich prezentuje. Užívateľské rozhranie pre vyhodnocovanie tohoto písania je natoľko jednoduché a zrozumiteľné, že aplikáciu môžu používať aj absolútni začiatočníci na počítači a informáciám porozumejú.

Okrem samotnej aplikácie hodnotím ako veľký prínos vyriešenie problému zachytávania a prekladu kláves na platforme Microsoft Windows 32-bit. Chybovosť funkcií `ToAsciiEx` a `ToUnicodeEx` je už dlho známa, no pri vývoji aplikácie sa mi nepodarilo nájsť riešenie, ktoré by toto správanie opravovalo. Pri testovaní aplikácie neboli zistené žiadne chyby spojené s prekladom kláves. To znamená, že použité riešenie by mohlo byť prvé, ktoré korektne pracuje s jazykovými rozloženiami krajín, ktoré používajú mŕtve klávesy a zároveň je schopné reagovať na zmenu tohoto rozloženia. Pre vývojárov z väčšiny krajín Európy to znamená možnosť vyvíjať aplikácie zamerané na odchytanie kláves bez chybového správania.

Veľmi zaujímavé sú aj výsledky získané aplikáciou, histogramy časových rozostupov medzi dvomi stlačenými klávesami jasne ukazujú vzťah medzi rovnomernosťou a kvalitou užívateľovho písania. Tieto výsledky som umiestnil do dodatku B, kde sú zobrazené kompletné štatistiky niekoľkých vybraných užívateľov.

Aplikáciu som sprístupnil na stiahnutie na stránke `http://smarttyper2000.weebly.com/`, kde je možné stiahnuť si binárne súbory aplikácie potrebné ku jej spusteniu, ale aj zdrojové súbory, ktoré poslúžia programátorom.

V budúcnosti by som rád na vývoji aplikácie pokračoval, núka sa veľké množstvo možností. Jednou z nich je spájanie úsekov textu, ktoré majú k sebe blízko časovo. Užívateľ by na základe toho mohol napríklad zistiť, že od 15:23 do 16:38 písal dlhý text, kde rýchlosť jeho písania bola pomalšia ako inokedy. Či už súčasť tejto aplikácie, alebo už novej, by mohli byť cvičenia na zdokonaľovanie písania. Užívateľovi sa ukáže predloha a túto predlohu bude musieť opisovať. Tým, že je známa predloha, aj výsledok, by bolo možné použiť presnejšie metriky chybovosti. Takisto by bolo možné rozšíriť charakteristiky písania napríklad o priemernú vzdialenosť medzi dvomi preklepmi. Kvôli značne špecifickým funkciám použitých pri preklade kláves je aplikácia plne funkčné len na 32-bitovej architektúre, do budúcnosti by som rád funkčnosť aplikácie rozšíril aj na 64-bitové platformy.

Literatúra

[1] Arif, A. a Stuerzlinger, W. *Analysis of Text Entry Performance Metrics.* Toronto, Ontario, Canada, 2009. 3 s.

[2] Barnes, L. *How to become expert in typewriting: A complete instructor designed especially for the Remington typewriter.* [b.m.]: A.J. Barnes, 1890.

[3] Canavan, J. *The Evolution of Malicious IRC Bots* [online]. 2005 [cit. 6. května 2012]. Dostupné na: <http://www.symantec.com/avcenter/reference/the.evolution.of.malicious.irc.bots.pdf>.

[4] Honeycutt, J., Cassel, P., Tweney, D. et al. *Windows 95: Nastavení protředí a práce s registry.* 1. vyd. Smetanova 3, Brno, PSČ 602 00: UNIS publishing, 1996. ISBN 0-7897-0725X.

[5] Moreau, M.-A. *Writing Keyloggers* [online]. This page was last modified on 20 September 2008, at 21:30. [cit. 6. května 2012]. Dostupné na: <http://legacy.docdroppers.org/wiki/index.php?title=Writing_Keyloggers>.

[6] Neugebauer, T. *Profesionálem v administrative: učebnice, která váš naučí: psaní deseti prsty ...* 2. vyd. Olomouc: Rubico, 2004. 210 s. Knížka pro každého. ISBN 80-7346-016-5.

[7] Neugebauer, T. *Psaní na počítači - profesionální ovládání klávesnice* [online]. únor 2008 [cit. 6. května 2012]. Dostupné na: <http://www.psani-vsemi-deseti.cz/2045/psani_na_pocitaci_profes_01.html>.

[8] Parkinson, R. *The Dvorak Simplified Keyboard: Forty Years of Frustration* [online]. 1972, Last updated: 2003/06/14 00:58:40 [cit. 6. května 2012]. Dostupné na: <http://infohost.nmt.edu/ shipman/ergo/parkinson.html>.

[9] Petzold, C. *Programování ve Windows.* 1. vyd. Hornocholupická 22,143 00, Praha 4: Computer Press, 1999. ISBN 80-7226-206-8.

[10] Richter, J. *Windows pro pokrocilé a experty: Architektura 32bitových systému Windows 95 a Windows NT.* 1. vyd. Hornocholupická 22,143 00, Praha 4: Computer Press, 1997. ISBN 80-85896-89-3.

[11] WWW stránky. *About Keyboard Input* [online]. Build date: 3/6/2012 [cit. 6. května 2012]. Dostupné na: <http://msdn.microsoft.com/en-us/library/windows/desktop/ms646267(v=vs.85).aspx>.

[12] WWW STRÁNKY. *Create and Embed an Application Manifest (UAC)* [online]. [cit. 6. května 2012]. Dostupné na: <http://msdn.microsoft.com/en-us/library/bb756929.aspx>.

[13] WWW STRÁNKY. *Google Visualization API* [online]. [cit. 6. května 2012]. Dostupné na: <http://code.google.com/apis/chart/interactive/docs/reference.html>.

[14] WWW STRÁNKY. *KBDLLHOOKSTRUCT structure* [online]. Build date: 2/3/2012 [cit. 6. května 2012]. Dostupné na: <http://msdn.microsoft.com/en-us/library/windows/desktop/ms644967(v=vs.85).aspx>.

[15] WWW STRÁNKY. *Language Identifier Constants and Strings* [online]. Build date: 3/6/2012 [cit. 6. května 2012]. Dostupné na: <http://msdn.microsoft.com/en-us/library/windows/desktop/dd318693(v=vs.85).aspx>.

[16] WWW STRÁNKY. *Manage the Programs Run at Windows Startup* [online]. Last modified: August 15, 2002 [cit. 6. května 2012]. Dostupné na: <http://www.pctools.com/guides/registry/detail/109/>.

[17] WWW STRÁNKY. *Mt.exe* [online]. Build date: 2/3/2012 [cit. 6. května 2012]. Dostupné na: <http://msdn.microsoft.com/en-us/library/windows/desktop/aa375649(v=vs.85).aspx>.

[18] WWW STRÁNKY. *QTimer Class Reference* [online]. [cit. 6. května 2012]. Dostupné na: <http://qt-project.org/doc/qt-4.8/QTimer.html>.

[19] WWW STRÁNKY. *ToAsciiEx Function* [online]. Build date: 3/6/2012 [cit. 6. května 2012]. Dostupné na: <http://msdn.microsoft.com/en-us/library/windows/desktop/ms646318(v=vs.85).aspx>.

[20] WWW STRÁNKY. *Virtual-Key Codes* [online]. [cit. 6. května 2012]. Dostupné na: <http://msdn.microsoft.com/en-us/library/ms927178.aspx>.

[21] WWW STRÁNKY. *Windows API List* [online]. Build date: 3/6/2012 [cit. 6. května 2012]. Dostupné na: <http://msdn.microsoft.com/en-us/library/ff818516(v=vs.85).aspx>.

[22] WWW STRÁNKY. *Windows Driver Kit (WDK)* [online]. [cit. 6. května 2012]. Dostupné na: <http://msdn.microsoft.com/en-us/windows/hardware/gg487428>.

[23] YAMADA, H. A historical study of typewriters and typing methods: From the position of planning Japanese parallels. *The Journal of Information Processing.* 1980, roč. 2. ISSN 1882-6652.

Dodatok A

Obsah CD

- Bakalárska práca v elektronickej podobe vrátane zdrojových kódov systému LaTeX.
- Zdrojové kódy aplikácie Smart Typer 2000 vrátane programovej dokumentácie
- Binárne súbory aplikácie Smart Typer 2000
- Jednoduchý videonávod, ktorý ukazuje ako aplikáciu používať
- Plagát vo formáte A2 prezentujúci bakalársku prácu

Dodatok B

Výsledky užívateľského testovania

Táto príloha obsahuje výsledky užívateľského testovania. Nachádzajú sa tu kompletné štatistiky písania užívateľov. Užívatelia sú vyberaní rôznorodo, aby bolo jasne vidieť rozdiely medzi nimi.

B.1 Muž, 51 rokov, podnikateľ

Histogram časových rozostupov medzi dvomi stlačenými klávesami tohoto užívateľa už bol zverejnený v kapitole 2.1.1, preto ho nebudem uvádzať aj v tejto prílohe.

Zvyšné štatistiky tohoto užívateľa vyzerajú takto:

Štatistický údaj	Krátke odstavce	Stredne dlhé odstavce	Dlhé odstavce	Súhrn
Počet odstavcov	208	16	1	225
Počet slov	998	202	42	1242
Počet úderov do klávesnice	5094	1014	209	6317
Dĺžka slova	5.10	5.02	4.98	5.03
Rýchlosť	49.19	35.81	43.29	42.76
Chybovosť	0.04	0.06	0	0.05
Najčastejšie preklepy	o->i, d->s, i->o	e->a, ,->?, o->v	—	o->i, e->a, ,->?

Tabuľka B.1: Štatistiky užívateľa

Či už zo spomínaného histogramu, alebo zo samotných štatistík, je jasné, že užívateľ je začiatočník pri práci na počítači. Jeho rýchlosť je veľmi nízka a má pomerne vysokú chybovosť. Jeho preklepy sú typické pre užívateľov, ktorý neovládajú žiadnu techniku písania - časté mýlenie susedných písmen.

B.2 Žena, 52 rokov, pracovníčka v administratíve

Histogram časových rozostupov medzi dvomi stlačenými klávesami tohoto užívateľa už bol zverejnený v kapitole 2.1.2, preto ho nebudem uvádzať aj v tejto prílohe.

Zvyšné štatistiky tohoto užívateľa vyzerajú takto: Štatistiky tohoto užívateľa v skutku

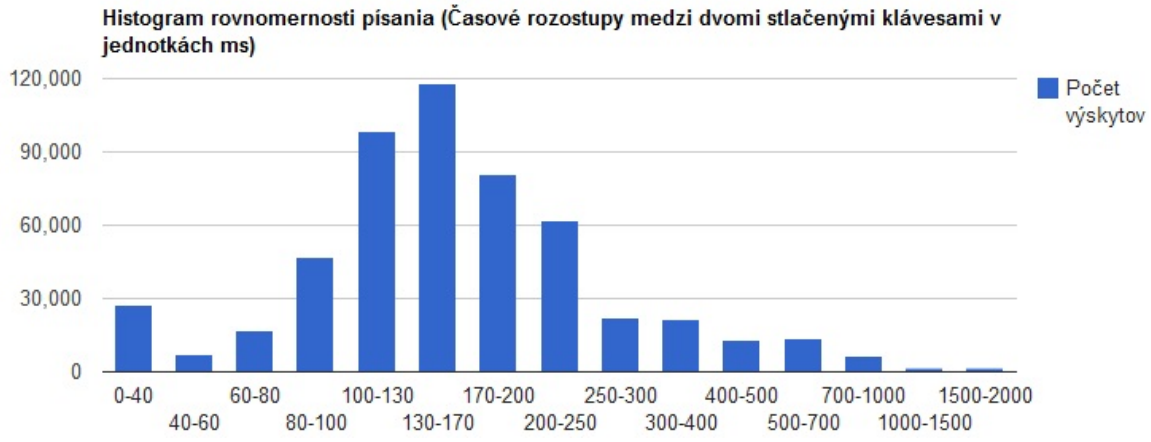
Štatistický údaj	Krátke odstavce	Stredne dlhé odstavce	Dlhé odstavce	Súhrn
Počet odstavcov	10	10	24	44
Počet slov	65	152	836	1053
Počet úderov do klávesnice	325	832	4691	5848
Dĺžka slova	5.00	5.47	5.61	5.36
Rýchlosť	79.21	72.31	66.78	72.77
Chybovosť	0.02	0.02	0.02	0.02
Najčastejšie preklepy	—	—	l->o, d->z, á->á	l->o, d->z, á->á

Tabuľka B.2: Štatistiky užívateľa

odpovedajú jeho schopnostiam. Je jasné, že písanie ovláda na veľmi vysokej úrovni, rýchlosť je síce priemerná, no užívateľ robí minimum chýb. Z jeho histogramu je na prvý pohľad viditeľná vysoká pravidelnosť písania.

B.3 Žena, 22 rokov, študentka práva

Histogram časových rozostupov medzi dvomi stlačenými klávesami vyzerá nasledovne:



Obr. B.1: Histogram časových rozostupov medzi dvomi stlačenými klávesami

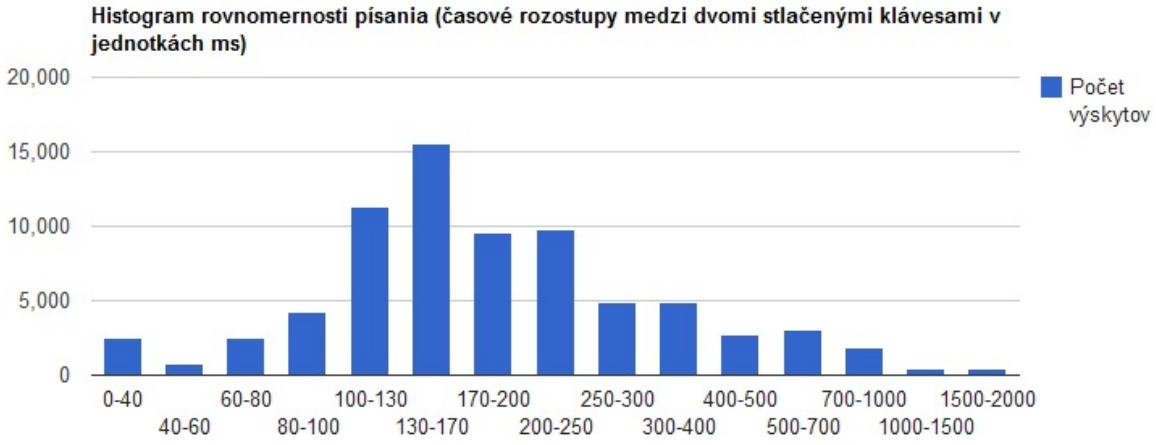
Zvyšné štatistiky tohoto užívateľa vyzerajú takto:

Štatistický údaj	Krátke odstavce	Stredne dlhé odstavce	Dlhé odstavce	Súhrn
Počet odstavcov	5533	2277	1475	9285
Počet slov	32312	32502	43716	108530
Počet úderov do klávesnice	143561	161536	230071	535168
Dĺžka slova	4.44	4.97	5.26	4.89
Rýchlosť	90.97	74.24	69.65	78.29
Chybovosť	0.07	0.08	0.09	0.08
Najčastejšie preklepy	D->:,l->k,i->o,	D->:,p->o,e->n	p->o, e->n, z->y	p->o, D->:, e->n

Tabuľka B.3: Štatistiky užívateľa

Zo štatistík toho užívateľa je možné napríklad odvodiť to, že užívateľ píše na klávesnici rýchlo, no nepravidelne, a z toho vyplýva aj väčšia chybovosť. Časté preklepy D->: indikujú časté prepisovanie pri používaní smailíkov. Preklep z->y môže napríklad znamenať problémy pri prepínaní medzi QWERTY a QWERTZ klávesnicou.

B.4 Žena, 21 rokov, študentka biotechnológie



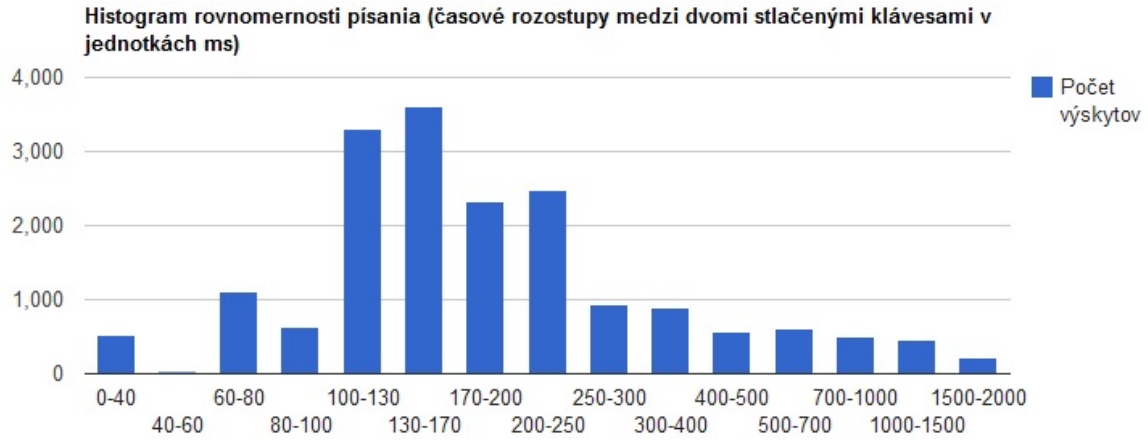
Obr. B.2: Histogram časových rozostupov medzi dvomi stlačenými klávesami

Štatistický údaj	Krátke odstavce	Stredne dlhé odstavce	Dlhé odstavce	Súhrn
Počet odstavcov	1204	312	109	1625
Počet slov	6577	4358	2947	13882
Počet úderov do klávesnice	33905	24208	16285	74398
Dĺžka slova	5.16	5.55	5.53	5.41
Rýchlosť	66.51	64.08	61.62	64.07
Chybovosť	0.06	0.07	0.08	0.07
Najčastejšie preklepy	y->z, a->e, z->y	l->a, á->í, e->i	o->p, z->y, s->d	á->í,y->z,a->e

Tabuľka B.4: Štatistiky užívateľa

Pri bližšej analýze štatistík tohoto užívateľa som zistil, že užívateľ nepoužíva žiadnu konkrétnu techniku písania, jeho rýchlosť aj rovnomernosť písania sú priemerné a z toho pramení aj zvýšená chybovosť. Zaujímavá je analýza preklepov, pri ktorých je najčastejšia kombinácia á->í,čo indikuje, že užívateľ implicitne diakritiku nepoužíva, no keď musí, tak v jej používaní nie je príliš zbehlý. Tak isto je zrejmé, že užívateľ používa aktívne QWERTY aj QWERTZ klávesnicu, a pri ich prepínaní má problém si zvyknúť na rozdielnu polohu Y a Z.

B.5 Muž, 22 rokov, študent informatiky



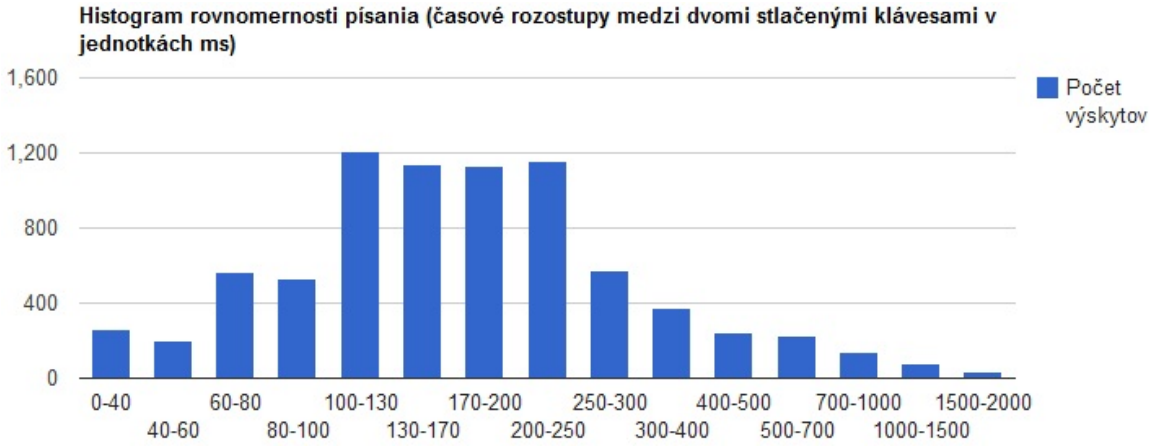
Obr. B.3: Histogram časových rozostupov medzi dvomi stlačenými klávesami

Je vidieť, že tento užívateľ píše oproti ostatným aj mierne rovnomernejšie a dokonca aj menej chybovo. Podľa jeho histogramu je však vidieť, že zrejme nepíše žiadnou technikou písania, a je to samouk. Preto sú s najväčšou pravdepodobnosťou jeho lepšie výsledky spojené s jeho študijným oborom - informatikou.

Štatistický údaj	Krátke odstavce	Stredne dlhé odstavce	Dlhé odstavce	Súhrn
Počet odstavcov	456	75	10	541
Počet slov	2294	1021	250	3565
Počet úderov do klávesnice	11003	5490	1464	17957
Dĺžka slova	4.80	5.38	5.86	5.34
Rýchlosť	76.11	61.35	56.23	64.56
Chybovosť	0.05	0.07	0.05	0.06
Najčastejšie preklepy	D->a,n->m, p->b	t->p, e->e, ,->m	:->a, s->S, š->c	b->v, s->d, t->p

Tabuľka B.5: Štatistiky užívateľa

B.6 Muž, 20 rokov, študent politológie



Obr. B.4: Histogram časových rozostupov medzi dvomi stlačenými klávesami

Štatistický údaj	Krátke odstavce	Stredne dlhé odstavce	Dlhé odstavce	Súhrn
Počet odstavcov	97	40	14	151
Počet slov	562	561	405	1528
Počet úderov do klávesnice	2527	3024	2234	7785
Dĺžka slova	4.50	5.39	5.52	5.13
Rýchlosť	87.71	68.28	62.46	72.82
Chybovosť	0.09	0.10	0.11	0.10
Najčastejšie preklepy	š->č,m->n, á->í	u->i, l->n, j->a	e->p,n->m,b->v	u->i, t->j, á->í

Tabuľka B.6: Štatistiky užívateľa

Na prvý pohľad do štatistík sú viditeľné vysoké rýchlosti písania, pri bližšom skúmaní je však jasné, že užívateľ píše len priemernou technikou, a vysoká rýchlosť je vyvážený veľmi vysokou chybovosťou, často krát pri písani písmen obsahujúcich diakritiku.