FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITA KOMENSKÉHO V BRATISLAVE

Počítač učiteľ čítania

Experimentálna implementácia

DIPLOMOVÁ PRÁCA

Martin Malár 2007

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
UNIVERZITA KOMENSKÉHO V BRATISLAVE
KATEDRA INFORMATIKY

Počítač učiteľ čítania

Experimentálna implementácia

DIPLOMOVÁ PRÁCA

Odbor informatika

Autor: Martin Malár
Školiteľ: RNDr. Marek Nagy

BRATISLAVA 2007

Čestne vyhlasujem, že diplomovú prácu som vypracoval samostatne, s využitím literatúry a zdrojov uvedených v závere práce.

V Bratislave, máj 2007

Martin Malár

Pod'akovanie
Touto cestou vyslovujem svoje pod'akovanie pánovi RNDr. Marekovi Nagyovi za odborné vedenie práce, poskytnutie literatúry, cenné rady a pomoc pri vypracovaní mojej diplomovej práce.

Abstrakt

Motiváciou pre túto prácu je využitie techník rozpoznávania reči na implementáciu didaktickej pomôcky na výučbu čítania pre deti v mladšom školskom veku s využitím už existujúcich nástrojov na rozpoznávanie reči. Taktiež je cieľom práce preskúmanie možností zaradenia tejto pomôcky do edukačného procesu, uskutočnenie experimentu na jej otestovanie a načrtnutie možností ďalšieho rozšírenia.

Kľúčové slová: rozpoznávanie reči gcompris linux

Obsah

Zoznam obrázkov

Úvod

Človek sa už od nepamäti snaží uľahčiť si svoj život. Príchod počítačov a ich rýchly pokrok umožnil ich využitie v rôznych oblastiach a odvetviach a ponúka široké možnosti ich uplatnenia. Počítače nám dnes pomáhajú pri práci aj pri štúdiu, riadia veľké továrne, vojenskú techniku a vesmírne lode, sú zdrojom zábavy, zdieľania informácií a ľudom na celom svete umožňujú vzájomne komunikovať. Prirodzená ľudská reč už dávno nie je jediným komunikačným prostriedkom, avšak stále ostáva tým najdôležitejším. Skúsme sa zamyslieť nad tým, či by počítač mohol byť vhodným nástrojom na výučbu reči a čítania pre naše deti, pokúsme sa preskúmať jeho možnosti a tento cieľ realizovať. Komunikácia s počítačom prirodzeným ľudským jazykom je jedným z najstarších snov z oblasti počítačov, o ktorého naplnenie sa ľudstvo snaží už od počiatku éry počítačov. A treba podotknúť, že nie neúspešne, hoci ešte stále sme veľmi ďaleko od stavu, aký by sme si predstavovali.

Z viacerých problémov, týkajúcich sa komunikácie s počítačom, nás bude v tejto práci zaujímať hlavne rozpoznávanie reči. Rozpoznávanie reči je tá časť vzájomnej komunikácie medzi človekom a počítačom, kedy počítač analyzuje hlasovú vzorku a snaží sa uhádnuť jednotlivé slová. Medzi ostatné problémy v tejto oblasti patria napr. aj spracovanie prijatého signálu, pochopenie významu rozpoznaných slov, príprava odpovede či vykonanie príslušnej akcie alebo syntéza reči. V dnešnej dobe sú už pomerne dostupné programy schopné rozpoznávania reči v nejakej jej forme, napriek tomu problém rozpoznávania reči nie je ešte ani zďaleka vyriešený. Problémy, ktoré úspešné rozpoznávanie reči sťažujú, vyplývajú predovšetkým z matematickej podstaty počítačov. Každý človek rozpráva trochu inak a má iný hlas, ešte aj slová vyslovené tou istou osobou sú vždy vyslovené trochu rôzne, odlišným tónom, inou hlasitosťou, s mierne rozdielnou intonáci-

ou. K tomu prispejú ďalšie vplyvy prostredia (napr. šum alebo hluk) a keď si to počítač preloží do svojej reči čísel, tá istá veta môže byť reprezentovaná diametrálne odlišnými údajmi, čo vedie k veľkej zložitosti algoritmov potrebných na rozpoznanie slov a k nižšej úspešnosti ich správneho rozpoznania.

Táto práca zachytáva hneď niekoľko pohľadov na problémy, ktoré sme uviedli vyššie. Našim cieľom bude preskúmať možnosti využitia počítača pri výučbe čítania s využitím rozpoznávania reči implementáciou programu, schopného kontrolovať správnosť výslovnosti rečníka, čítajúceho zobrazený text. Prvý pohľad na túto prácu je teda implementačný, keďže sa jedná o naprogramovanie softvérového produktu. Iný pohľad na túto prácu je problém rozpoznávania reči s tým, že tento problém sa nebudeme snažiť vyriešiť teoreticky, ale prakticky tým, že sa pokúsime využiť už existujúce nástroje na rozpoznávanie reči a prispôsobiť si ich tak, aby vyhovovali nášmu zámeru. Posledný pohľad je pohľad pedagogický, kedy my tento náš nástroj vyskúšame v praxi a budeme skúmať, čo a ako by sa dalo vylepšiť a čo bude treba na to, aby sa tento spôsob výučby mohol úspešne zaviesť do edukačného procesu.

V prvej kapitole si pohovoríme o teoretických základoch rozpoznávania reči a vysvetlíme si, akým spôsobom rozpoznávanie funguje. V kapitole druhej uvedieme detaily týkajúce sa samotného programu, jeho filozofiu, implementáciu a rozoberieme jeho funkcie. Taktiež načrtneme všetko to, čo by sa mohlo urobiť v budúcnosti alebo to, čo presahuje rámec tejto práce a čo by tento program mal alebo mohol obsahovať. Tretia kapitola pojednáva o pedagogickom aspekte celej práce, obsahuje návrhy a postupy pre použitie programu v praxi alebo pri zavádzaní programu do edukačného procesu pre rôzne vekové kategórie a taktiež v nej uvádzame výsledky našich experimentov z praxe. V závere potom zosumarizujeme výsledky, teda všetko to, k čomu sme počas tejto práce prišli a taktiež pouvažujeme nad ďalším smerovaním tohto projektu.

KAPITOLA PRVÁ

ROZPOZNÁVANIE REČI

V tejto kapitole budeme diskutovať o teoretických základoch rozpoznávania reči a bližšie si vysvetlíme, ako to celé prebieha. Budeme sa zaoberať predovšetkým samotným rozpoznávaním reči, ostatné časti problematiky, ako sú napr. spracovanie akustického signálu, syntéza reči a pod., presahujú rámec tejto práce a preto sa nimi nebudeme podrobne zaoberať[1]. Prejdeme cez niekoľko rôznych prístupov k rozpoznávaniu reči a stručne si vysvetlíme ich princípy. V oddiele 1.1 si vysvetlíme pojem *fonetická abeceda*, v oddiele 1.2 prediskutujeme algoritmy na rozpoznávanie reči, založené na princípoch dynamického programovania a štatistických metódach a taktiež budeme hovoriť o rozdieloch medzi rozpoznávaním izolovaných slov a súvislej reči, napokon v oddiele 1.4 sa podrobnejšie pozrieme na Skryté Markovove modely, techniku, ktorá je využitá aj pri implementácii projektu k tejto práci.

1.1 Fonetická abeceda

Tak ako pri písaní textu sa používa abeceda, ktorá nám rozdelí celé slová na menšie celky – písmená (grafémy), tak aj pri rozprávaní môžeme používať abecedu, ktorá vyslovené slová rozdelí na menšie časti, ktoré nazveme *fonémy*. Fonémy sú vlastne reprezentáciou zvuku a sú často označované za najmenšiu jednotku reči. Zmena časti slova z jednej fonémy na inú môže vyprodukovať úplne iné slovo alebo zmysel. Vo svetových jazykoch sa bežne používa od 12 do okolo 60 foném. Fonémy ďalej tvoria slabiky a slabiky tvoria slová.

[1] Podrobnejšie informácie ohľadom daných problematík čitateľ nájde v [1].

Koartikulácia je jav, kedy sa fonéma môže meniť podľa kontextu, v ktorom je vyslovená. Konkrétna akustická realizácia danej fonémy potom závisí napr. od predchádzajúcej a nasledujúcej fonémy, rýchlosti rozprávania, intonácie a pod. Fonéma použitá v rôznych slovách môže byť reprezentovaná iným zvukom v závislosti na koartikulácii v jeho fonetickom okolí. Takýto variant fonémy nazývame *fón* a všetky rôzne fóny jednej fonémy nazývame *alofóny*. Rôzne alofóny sú použité podľa fonetického kontextu a môžu napr. vymedzovať hranice slova alebo slabiky.

Vo všeobecnosti je reč neharmonický, nestacionárny signál, a preto je len veľmi ťažko modelovateľná. Z mnohých pokusov však vyplýva, že počas krátkeho časového intervalu môžeme reč považovať za stacionárnu. Na fonémy, resp. na jej alofóny, sa môžeme pozerať ako na tento kus stacionárnej informácie, ktorá nám môže pomáhať nielen pri klasifikácii slov, pri ich rozpoznávaní, ale aj pri popise ich správnej výslovnosti. International Phonetic Alphabet (IPA)[2] vznikla ako výsledok dlhoročnej snahy jazykovedcov na celom svete. Jedná sa o systém fonetickej notácie, ktorej cieľom je poskytnúť štandartný a jednotný spôsob reprezentácie akéhokoľvek zvuku v ktoromkoľvek jazyku. Vo svojej základnej nerozšírenej forme obsahuje približne 107 základných symbolov a 55 modifikátorov.

Fonetická transkripcia je jednoznačný spôsob zápisu zvukov hovorenej reči za pomoci fonetickej abecedy, pričom IPA je pre tento účel tou najvšeobecnejšou abecedou. Výber vhodnej fonetickej abecedy závisí od toho, nakoľko presný popis chceme vytvárať – abeceda foném pre voľnejší popis alebo abeceda alofónov pre popis detailnejší. Tento popis sa používa na prípravu rozpoznávacích alebo trénovacích vzoriek, ktoré sa potom používajú k trénovaniu rozpoznávačov alebo na testovanie úspešnosti rozpoznávania slov.

1.2 Zakladné metódy

Poďme sa teraz pozrieť na rôzne prístupy k problémom rozpoznávania reči a stručne si prejdime metódy, ktoré sa tieto problémy snažia riešiť. V úvode sme

[2] z angl. *Medzinárodná fonetická abeceda*.

načrtli niektoré problémy, ktoré majú vplyv na obtiažnosť rozpoznávania. Josef Psutka vo svojej knihe [1] uvádza tieto tri hlavné problémy:

- Rozdielne osoby majú rozdielne hlasy. Tieto rozdiely sú spôsobené rôznym tvarom hlasového ústrojenstva a odlišným spôsobom artikulácie. Medzi ďalšie dôvody patria napr. farba hlasu, prízvuk atď.
- Hlas rečníka môže byť v rôznych situáciách rozdielny v závislosti na rýchlosti reči, jej hlasitosti alebo v závislosti od duševného a zdravotného stavu rečníka (rozčúlenie, nachladnutie, zachrípnutie, atď.). Navyše, aj bez týchto faktorov je takmer nemožné, aby jeden človek vyslovil to isté slovo úplne rovnako a rovnaké slovo vyslovené v rôznych kontextoch môže znieť rozdielne – koartikulácia.
- Hlučné prostredie má taktiež vplyv na výsledok rozpoznávania, napr. šum, hluk, rozhovor v pozadí atď., tým, že sťažuje identifikáciu začiatkov a koncov slov, modifikuje celý signál a zakrýva jeho charakteristické črty a pod.

Problémov spojených s rozpoznávaním hovoreného slova je samozrejme omnoho viac, spomenuté problémy sú ale tie najhlavnejšie. Bolo navrhnutých veľa prístupov k týmto problémom, z ktorých sa stručne pozrieme na metódu *aplikácie dynamického programovania* a v oddiele 1.4 si podrobnejšie prejdeme metódu nazvanú *skryté Markovove modely*.

V závislosti od úlohy, pre ktorú sme sa rozhodli použiť rozpoznávanie reči, pripadajú do úvahy dva spôsoby, akým budeme zvukový signál rozpoznávať. V niektorých prípadoch nám bude stačiť rozpoznávať slová z konečnej množiny slov, veľkej alebo malej v závislosti od rozsahu úlohy, pričom tieto slová budeme dostávať "po jednom", v tomto prípade sa bude jednať o rozpoznávanie izolovaných slov. Ako si neskôr ukážeme, naša množina slov nemusí byť nutne konečná v prípade, že budeme rozpoznávať menšie jednotky ako slová[3] a slová z nich budeme dostávať zreťazovaním. Aplikácia dynamického programovania alebo metóda skrytých Markovových modelov sú metódy, ktoré sa používajú predovšetkým na rozpoznávanie izolovaných slov.

[3] napr. fonémy, o ktorých sme diskutovali v oddiele 1.1.

Druhým spôsobom, akým môžeme rozpoznávať zvukový signál, je snažiť sa na neho pozrieť ako na súvislú reč. Rozpoznávanie súvislej reči so sebou prináša ďalšie problémy, s ktorými sa pri rozpoznávaní izolovaných slov nestretávame tak často. Jedným z tých hlavných je určenie hraníc jednotlivých slov, nakoľko pri plynulej reči sa jednotlivé slová zlievajú do súvislého zvukového signálu, v ktorom chýbajú pauzy ticha, ktoré by mohli identifikovať začiatky a konce slov. Napríklad táto jednoduchá veta: "Kam ideš?", obsahujúca dve slová, vyslovená plynule sa do hovorovej reči premietne ako tri slabiky: "Ka-mi-deš". Pauza, ktorá je v napísanom texte charakterizovaná medzerou, pri vyslovení vety zmizne.

Medzi dôvody, vedúce k skúmaniu rozpoznávania plynulej súvislej reči, patria snaha o plynulú komunikáciu s počítačom, pretože pre ľudí je prirodzené rozprávať plynule, ďalej potom rýchlosť komunikácie, ale aj plynulosť. Plynulá reč zvyšuje rýchlosť, ktorou údaje vstupujú do počítača a tým odľahčuje rozpoznávač, ktorý potom potrebuje prepočítať menšie množstvo dát, čím zrýchluje celý proces rozpoznávania. Nevýhodou v tomto smere sú potom zložitejšie algoritmy, potrebné na rozpoznanie súvislej reči.

Existujú rôzne algoritmy zaoberajúce sa riešením problémov okolo rozpoznávania súvislej reči, odvodené napr. od algoritmu založenom na dynamickom programovaní, o ktorom budeme písať v oddiele 1.3 v súvislosti s rozpoznávaním izolovaných slov. Iný prístup ďalej rozvíja štatistickú metódu skrytých Markovových modelov a ktorý je použiteľný ako na rozpoznávanie izolovaných slov, tak aj na rozpoznávanie súvislej reči a je použitý aj pri implementácii tohto projektu.

1.3 Aplikácia dynamického programovania

Metóda, o ktorej budeme hovoriť v tomto oddiele, patrí do skupiny metód, ktoré pri klasifikácii vezmú slovo ako celok a to sa snažia zaradiť do takej triedy slov, ktorej vzorový obraz má *najmenšiu vzdialenosť* od vstupného slova. Hlavným problémom pri takýchto algoritmoch je určenie vzájomnej vzdialenosti dvoch obrazov slov. Táto metóda využíva techniku dynamického programovania, pomocou ktorej vypočítava vzdialenosti medzi obrazmi. Názov metódy nie je odvodený od spôsobu samotného rozpoznávania, ale od spôsobu určovania vzdialenosti.

Rôzne vyslovenia toho istého slova sa môžu výrazne líšiť v časovom členení, čo má za následok rôzne celkové dĺžky slova, ale aj časový nepomer medzi dĺžkami častí jednotlivých slov, ako sú fonémy, hlásky, ale aj slabiky. Toto nelineárne kolísanie môže negatívne ovplyvňovať úspešné rozpoznanie slova. Problém rôznej dĺžky slov sa dá vyriešiť lineárnou časovou normalizáciou, tá však nedokáže odstrániť vnútorné kolísanie v slovách. Na to sa môže použiť nelineárna časová normalizácia s DTW[4] funkciou, ktorá eliminuje časové rozdiely vo vnútri slov tak, aby bola dosiahnutá maximálna zhoda s druhým obrazom.

Predpokladajme, že rečový signál máme spracovaný do postupnosti *pozorovaní*, resp. postupnosti vektorov pozorovaní. Vzťah (1.1) označuje obraz testovaného slova, vzťah (1.2) označuje obraz vzorového slova.

$$\mathbf{A} = \{\, a(1), a(2), \ldots, a(n), \ldots, a(I) \,\} \tag{1.1}$$

$$\mathbf{B} = \{\, b(1), b(2), \ldots, b(m), \ldots, b(J) \,\} \tag{1.2}$$

Keď sa na tieto dva vektory vektorov pozrieme ako na dvojrozmerný priestor, algoritmus s funkciou DTW hľadá optimálnu cestu ψ v rovine (m, n),

$$m = \psi(n) \tag{1.3}$$

ktorá minimalizuje výstupy funkcie $\mathbf{D}$. Funkcia $\mathbf{D}$ počíta celkovú vzdialenosť medzi obrazmi $\mathbf{A}$ a $\mathbf{B}$ a je definovaná vzťahom (1.4),

$$\mathbf{D}(\mathbf{A}, \mathbf{B}) = \sum_{n=1}^{I} \hat{\mathbf{d}}[\mathbf{a}(n), \mathbf{b}(\psi(n))] \tag{1.4}$$

pričom $\hat{\mathbf{d}}[\mathbf{a}(n), \mathbf{b}(\psi(n))]$ je lokálna vzdialenosť medzi n-tým vektorom pozorovaní testovaného slova a $\psi(n)$-tým vektorom pozorovaní referenčného slova.

Z rovnice (1.3) vyplýva, že hľadanú optimálnu cestu vieme vyjadriť jednoduchým funkčným vzťahom medzi m a n, my si však zavedieme všeobecnú časovú

[4] z angl. *dynamic time warping*.

premennú k a obe časové premenné m a n vyjadríme ako funkciu k,

$$\begin{aligned} n &= i(k), \quad k = 1, \ldots, K, \\ m &= j(k), \quad k = 1, \ldots, K, \end{aligned} \tag{1.5}$$

kde K je dĺžka všeobecnej časovej osi pre dané porovnanie obrazov $\mathbf{A}$ a $\mathbf{B}$.

Zo štruktúry rečových obrazov na časovej osi vyplývajú podmienky, ktoré musíme rešpektovať pri hľadaní optimálnej cesty funkcie DTW v rovine (n, m).

Pri spracovaní izolovaných slov je obmedzenie na hraničné body jednoduché, stačí funkciu DTW obmedziť podmienkami

$$\begin{aligned} i(1) &= 1, \quad j(1) = 1, \\ j(K) &= I, \quad j(K) = J. \end{aligned} \tag{1.6}$$

Aby sa zaručilo, že pri prechode funkcie DTW sa vyhneme nadmernej kompresii alebo expanzii časovej mierky, aplikujú sa na funkciu DTW obmedzenia na monotónnosť a spojitosť

$$\begin{aligned} 0 \leq i(k) - i(k-1) \leq I^*, \\ 0 \leq j(k) - j(k-1) \leq J^*. \end{aligned} \tag{1.7}$$

V praxi sa používajú hodnoty $I^*, J^* = 1, 2, 3$. Ak pripustíme hodnoty I^* alebo J^* väčšie ako 1, znamená to, že funkcia DTW môže pri porovnávaní niektoré segmenty vynechať.

Všeobecný tvar na určenie skutočnej minimálnej vzdialenosti medzi dvoma obrazmi $\mathbf{A}$ a $\mathbf{B}$ sa dá vyjadriť vzťahom

$$\mathbf{D}(\mathbf{A}, \mathbf{B}) = \min_{\{i(k), j(k), K\}} \left[\frac{\sum_{k=1}^{K} d\left[i(j), j(k), K\right] \hat{W}(k)}{N(\hat{W})} \right], \tag{1.8}$$

kde $d\left[i(j), j(k)\right]$ je lokálna vzdialenosť medzi $n = i(k)$-tym mikrosegmentom testovaného obrazu $\mathbf{A}$ a $m = j(k)$-tym mikrosegmentom referenčného obrazu $\mathbf{B}$, $\hat{W}(k)$ je hodnota váhovej funkcie pre k-ty úsek funkcie DTW a $N(\hat{W})$ je normalizačný faktor, ktorý je funkciou váhovej funkcie.

Podrobnejšie informácie ohľadom tohto algoritmu čitateľ nájde v [1].

1.4 Skryté Markovove modely

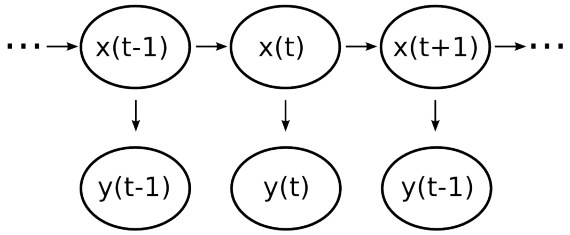
Metóda *skrytých Markovových modelov* je štatistická metóda, využívajúca výpočtové modely z teórie formálnych jazykov a automatov[5]. Na modelovanie reči touto metódou predpokladajme, že pri vytváraní reči sa naše hlasové ústrojenstvo nachádza v určitom časovom momente v jednej z konečného počtu artikulačných konfigurácií. Počas tohto krátkeho časového okamžiku (hovorme mu *mikrosegment*) naše hlasivky vyprodukujú krátky zvukový signál, ktorý závisí od aktuálneho stavu hlasového ústrojenstva a ktorý môžeme popísať určitými spektrálnymi charakteristikami. Počet týchto spektrálnych charakteristík nie je konečný, no pre naše potreby budeme predpokladať, že konečný je. Túto konečnosť dosiahneme *vektorovou kvantizáciou*, technikou, pomocou ktorej vytvoríme tzv. *kódovú knihu* typových spektrálnych vzorov. Každú spektrálnu charakteristiku potom nahradíme indexom "najbližšieho" spektrálneho vzoru z kódovej knihy.

Poďme sa teraz pozrieť, ako taký Markovov model vyzerá a ako v ňom využívame filozofiu konečného počtu artikulačných konfigurácií. Markovov model je konečný automat, ktorý má konečný počet stavov. Tieto stavy zodpovedajú stavom, v ktorých sa môže nachádzať hlasové ústrojenstvo počas vyslovenia slova. Medzi stavmi sú prechody, pričom každý prechod má priradenú pravdepodobnosť, na základe ktorej automat mení stavy. V regulárnom Markovovom modele je každý stav priamo viditeľný pozorovateľom, preto sú pravdepodobnosti prechodov len akýmisi parametrami. V skrytom Markovovom modele stavy nie sú viditeľné, ale sú viditeľné náhodné premenné, ktoré sú týmito stavmi ovplyvňované[6].

Na obrázku 1.1 je zobrazená všeobecná architektúra skrytého Markovovho modelu. Každý ovál znázorňuje náhodnú premennú, ktorá môže nadobúdať rôzne hodnoty. Náhodná premenná $x(t)$ je skrytá premenná v čase t. Náhodná premenná $y(t)$ je hodnota pozorovanej premennej v čase t. Hodnota skrytej náhodnej premennej $x(t)$ v čase t závisí iba od hodnoty skrytej premennej $x(t-1)$ v čase $t-1$. Podobne, hodnota premennej $y(t)$ závisí iba od hodnoty skrytej premennej

[5]Budeme predpokladať, že čitateľ má aspoň zbežné znalosti z tejto problematiky, v prípade potreby si môže nájsť viac informácií v [2].

[6]Preto sa tomuto modelu hovorí skrytý, lebo pozorovateľ vidí len výstupy náhodných funkcií, ale nie samotné stavy.



Obrázok 1.1: Architektúra Skrytého Markovovho modelu.

$x(t)$, oboje v čase t.

Konštrukcia klasifikátora pre skryté Markovove modely je založená na modelovaní rečového signálu pomocou Markovovho procesu. Pri tomto procese sú generované dve navzájom súvisiace časové postupnosti náhodných premenných, a to *podporný Markovov reťazec*, ktorý je postupnosťou konečného počtu stavov a reťazec konečného počtu *spektrálnych vzorov*. Pre jednotlivé spektrálne vzory sú vytvorené náhodné funkcie, ktoré pravdepodobnostne ohodnocujú vzťah týchto vzorov ku všetkým stavom. Predpokladáme, že v diskrétnych časových okamihoch je proces v jedinom stave a dá sa pozorovať prostredníctvom náhodnej funkcie korešpondujúcej s náhodným stavom. Podporný Markovov reťazec potom mení stavy podľa svojej matice pravdepodobností prechodu. Pozorovateľ vidí len výstup náhodných funkcií a nemôže priamo pozorovať stavy podporného Markovovho reťazca.

1.4.1 Matematický základ metódy

Pozrime sa teraz podrobnejšie na Markovove modely a zadefinujme si ich. Skrytý Markovov model $\mathcal{G}$ je reprezentovaný konečnostavovým automatom a môžeme ho vyjadriť päticou

$$\mathcal{G} = (Q, V, \mathcal{N}, \mathcal{M}, \pi) \tag{1.9}$$

kde:

— $Q = \{ q_1, \ldots, q_N \}$ je konečná množina N stavov Markovovho modelu.

— $V = \{ v_1, \ldots, v_L \}$ je abeceda L výstupných symbolov vektorového kvantizéru, zodpovedajúca indexom príslušných spektrálnych vzorov v kódovej knihe $\mathcal{V}$.

— $\mathcal{N} = [n_{ij}]$ je *matica prechodov*. Jej prvky určujú, s akou pravdepodobnosťou prechádza systém zo stavu q_i v ľubovoľnom čase t do stavu q_{i+1} v čase $t + 1$, čo môžeme zapísať ako

$$n_{ij} = P\left(q(t+1) = q_j \mid q(t) = q_i\right), \quad 1 \leq i, j \leq N. \tag{1.10}$$

— $\mathcal{M} = [m_{jl}] = [m_j(l)]$ je *matica pravdepodobností generovaných vzorov*, ktorej prvky určujú, s akou pravdepodobnosťou je v ľubovoľnom čase t generovaná l-tá položka konečnej množiny spektrálnych vzorov, ak je systém v stave q_i. Teda platí

$$m_{jl} = m_j(l) = P(v(t) = v_l \mid q(t) = q_j), \quad 1 \leq j \leq N, 1 \leq l \leq L. \tag{1.11}$$

— $\pi = [\pi_i]$ je stĺpcový *vektor pravdepodobností počiatočného stavu*, pre ktorý platí

$$\pi_i = P(q(1) = q_i), \quad 1 \leq i \leq N. \tag{1.12}$$

Zaveďme ešte označenie λ pre množinu parametrov Markovovho modelu, ktorý pozostáva z prvkov matíc $\mathcal{N}$, $\mathcal{M}$ a vektoru π

$$\lambda = (\mathcal{N}, \mathcal{M}, \pi). \tag{1.13}$$

1.4.2 Stanovenie pravdepodobnosti modelu.

Jedným z najhlavnejších problémov pre skryté Markovove modely je určenie pravdepodobnosti, že danú postupnosť pozorovaní vygeneroval daný model. Označme $O = o_1, o_2, \ldots, o_T$ *postupnosť pozorovaní*, kde T je počet mikrosegmentov a kde o_t je pozorovanie v čase t. Našou úlohou bude pre daný model $\lambda = (\mathcal{N}, \mathcal{M}, \pi)$ určiť pravdepodobnosť $P = (O \mid \lambda)$. Na tento problém sa môžeme pozerať aj ako na to, nakoľko zodpovedá model postupnosti pozorovaní.

Najjednoduchší spôsob je prepočítanie všetkých postupností pozorovaní, ktoré majú dĺžku T (alebo inak povedané, ktoré pozostávajú z T mikrosegmentov). Vezmime si ľubovoľnú, ale pevne danú, postupnosť stavov

$$Q = q_1, q_2, \ldots, q_T$$

kde q_1 je počiatočný stav. Pravdepodobnosť pozorovania postupnosti O pre danú postupnosť stavov Q je

$$P(O \mid Q, \lambda) = \prod_{t=1}^{T} P(o_t \mid q_t, \lambda) \tag{1.14}$$

pričom je zabezpečená štatistická nezávislosť pozorovaní. Potom dostávame výraz

$$P(O \mid Q, \lambda) = m_{q_1}(o_1)\; m_{q_2}(o_2)\; \ldots\; m_{q_T}(o_T) \tag{1.15}$$

Pravdepodobnosť každej postupnosti stavov Q sa dá zapísať ako

$$P(Q \mid \lambda) = \pi_{q_1}\; n_{q_1 q_2}\; n_{q_2 q_3}\; \ldots\; n_{q_{T-1} q_T} \tag{1.16}$$

Spojená pravdepodobnosť pravdepodobností O a Q (teda keď sa skúma pravdepodobnosť modelu pre danú postupnosť pozorovaní O pri danej postupnosti stavov Q) sa rovná súčinu týchto pravdepodobností, teda

$$P(O, Q \mid \lambda) = P(O \mid Q, \lambda)\; P(Q \mid \lambda) \tag{1.17}$$

Pravdepodobnosť O sa pri danom modely vypočíta ako suma všetkých spojených pravdepodobností cez všetky možné postupnosti stavov

$$\begin{aligned} P(Q \mid \lambda) &= \sum_{\forall Q} P(O \mid Q, \lambda)\; P(Q \mid \lambda) \\ &= \sum_{q_1, q_2, \ldots, q_T} \pi_{q_1}\; m_{q_1}(o_1)\; n_{q_1 q_2}\; m_{q_2}(o_2)\; \ldots\; n_{q_{T-1} q_T}\; m_T(o_T) \end{aligned} \tag{1.18}$$

Interpretácia predchádzajúceho výrazu je nasledovná; na začiatku v čase $t = 1$ sme v stave q_1 s pravdepodobnosťou π_{q_1} a v tomto stave generujeme symbol o_1

s pravdepodobnosťou $m_{q_1}(o_1)$. Potom sa zmení čas z t na $t+1$ (čiže na $t=2$) a my zmeníme stav z q_1 na stav q_2 s pravdepodobnosťou $n_{q_1q_2}$ a generujeme symbol o_2 s pravdepodobnosťou $m_{q_2}(o_2)$. Tento proces opakujeme dovtedy, kým nespravíme posledný prechod zo stavu q_{T-1} do stavu q_T s pravdepodobnosťou $n_{q_{T-1}q_T}$ a vygenerujeme symbol o_T s pravdepodobnosťou $m_{q_T}(o_T)$.

Tento spôsob výpočtu pravdepodobnosti $P(O \mid \lambda)$ priamo podľa definície je príliš náročný. Výpočet obsahuje $2T.N^T$ operácií, pretože v každom časovom okamihu $t = 1, 2, \ldots, T$ je možné sa dostať do N rôznych stavov (teda existuje N^T možných postupností stavov) a v každej postupnosti je potrebných $2T$ operácií (presnejšie $(2T-1)N^T$ násobení a $N^T - 1$ sčítaní). Toto množstvo operácií je neprijateľné už aj pre malé hodnoty N a T, napríklad pre $N = 5$ stavov a $T = 100$ pozorovaní dostaneme približne 10^{72} operácií. Je potrebné pohľadať iný, efektívnejší spôsob výpočtu problému určenia pravdepodobnosti modelu. Taký spôsob existuje, nazýva sa *forward-backward algorithm* a využíva rekurzívny výpočet spredu alebo zozadu na generovanej postupnosti.

Pri výpočte spredu uvažujme premennú $\alpha(i)$, ktorá je definovaná nasledovne

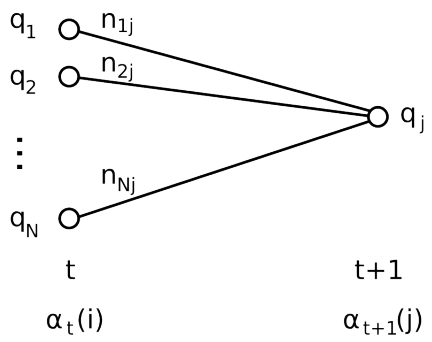
$$\alpha(i) = P(o_1 o_2 \ldots o_t, q(t) = q_i \mid \lambda) \tag{1.19}$$

Táto premenná $\alpha(i)$ je vlastne pravdepodobnosť generovania čiastkovej postupnosti pozorovaní $o_1, o_2, \ldots, o_t$ (až po čas t) a pozorovanie stavu q_i v čase t pre daný model λ. Hodnoty $\alpha_t(i)$ možno počítať rekurzívne:

1. Inicializácia
$$\alpha_1(i) = \pi_i \, m_i(o_i) \quad 1 \leq i \leq N. \tag{1.20}$$

2. Rekurzia pre $t = 1, 2, \ldots, T-1$
$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) \, n_{ij} \right] m_j(o_{t+1}) \quad 1 \leq j \leq N \tag{1.21}$$

3. Výsledná pravdepodobnosť
$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{1.22}$$



Obrázok 1.2: Ilustrácia výpočtu premennej $\alpha_t(i)$.

Krok 1) je inicializáciou pravdepodobností ako spojená pravdepodobnosť stavu q_i a počiatočného pozorovania o_1. Krok rekurzie, ktorý je jadrom výpočtu, je ilustrovaný na obrázku 1.2. Tento obrázok ukazuje, ako sa môžeme dostať v čase $t+1$ do stavu q_i z N možných stavov. Keďže je $\alpha_t(i)$ spojená pravdepodobnosť toho, že sme pozorovali $o_1 o_2 o_3 \ldots o_t$ a v čase t sme v stave q_i, potom je súčin $\alpha_t(i)\, n_{ij}$ spojenou pravdepodobnosťou toho, že sme pozorovali $o_1 o_2 \ldots o_t$ a v čase $t+1$ sme v stave q_j, kam sme sa dostali zo stavu q_i (kde sme boli v čase t). Suma tohto súčinu cez všetky možné stavy q_i, $i \leq i \leq N$, v čase t je pravdepodobnosťou stavu q_j v čase $t+1$, pričom sú obsiahnuté všetky predchádzajúce čiastkové postupnosti pozorovaní. Indukcia sa počíta pre všetky stavy v čase $t = 1, 2, \ldots, T-1$. Nakoniec, v kroku 3), získame hľadanú pravdepodobnosť $P(O \mid \lambda)$ ako sumu posledných premenných $\alpha_T(i)$

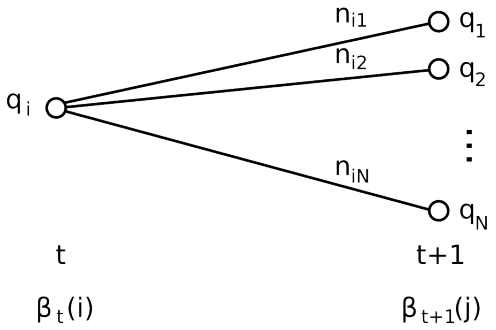
$$\alpha_T(i) = P(o_1 o_2 \ldots o_T, q_T = q_i \mid \lambda). \tag{1.23}$$

Ak spočítame počet operácií, potrebných pri výpočte

$$\alpha_T(i), 1 \leq t \leq T, 1 \leq j \leq N,$$

zistíme, že pre tento postup ich potrebujeme len N^2T (presnejšie $N(N+1)(T-1) + N$ násobení a $N(N-1)(T-1)$ sčítaní). Pre $N = 5$ a $T = 100$ teda potrebujeme len približne 3000 operácií oproti 10^{72} pri výpočte priamom.

Počítanie pravdepodobnosti spredu je založené na mriežkovej štruktúre. Podstata spočíva v tom, že je len N možných stavov (uzlov v časových slotoch mriež-



Obrázok 1.3: Ilustrácia výpočtu premennej $\beta_t(i)$.

ky) a všetky možné postupnosti stavov pôjdu cez tieto stavy, pričom nezáleží, aká dlhá je postupnosť pozorovaní. V čase $t = 1$ (prvý časový slot v mriežke) potrebujeme vypočítať hodnoty $\alpha_1(i), 1 \leq N \leq N$. V časoch $t = 2, 3, \ldots, T$ potrebujeme vypočítať už len hodnoty $\alpha_t(i), 1 \leq i \leq N$, kde každý výpočet zahŕňa len N predchádzajúcich hodnôt $\alpha_{t-1}(j)$, pretože do každého z N uzlov sa možno dostať len z tých istých N uzlov z prechádzajúceho časového slotu.

Podobne môžeme uvažovať o premennej $\beta_i(t)$ pre výpočet odzadu, definovanej ako pravdepodobnosť generovania čiastočnej postupnosti $\{o_{t+1}\, o_{t+2}\, \ldots\, o_T\}$ pri danom stave $q(t) = q_i$ a modele λ nasledovne

$$\beta_t(i) = P(o_{t+1} o_{t+2} \ldots o_T \mid q_t = q_i, \lambda). \tag{1.24}$$

1. Inicializácia:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N. \tag{1.25}$$

2. Rekurzia pre $t = T - 1, T - 2, \ldots, 1$

$$\beta_t(i) = \sum_{j=1}^{N} n_{ij}\, m_j(o_{t+1})\, \beta_{t+1}(j), \quad 1 \leq i \leq N. \tag{1.26}$$

3. Výsledná pravdepodobnosť

$$P(O \mid \lambda) = \sum_{i=1}^{N} \pi_i\, m_i(o_1)\, \beta_1(i). \tag{1.27}$$

Ďalším problémom, ktorým sa teraz budeme zaoberať, je určenie optimálnej (najpravdepodobnejšej) postupnosti stavov $Q = q_1, q_2, \ldots, q_T$, ktorá najlepšie "vystihuje" postupnosť pozorovaní $O = o_1, o_2, \ldots, o_T$ a model λ. Pri riešení tohto problému sa budeme snažiť odkryť skrytú časť modelu, teda nájsť správnu postupnosť stavov. Pri týchto modeloch neexistuje správna postupnosť, ale v praxi sa na základe rôznych kritérií snažíme riešiť tento problém v rámci možností. Existuje niekoľko optimalizačných kritérií, ktoré môžeme použiť, ale ich výber je silne závislý od účelu, na ktorý sa použije získaná postupnosť stavov. Typické použitie je štúdium štruktúry modelu či nájdenie optimálnej postupnosti stavov pri rozpoznávaní súvislej reči.

Pomocou premenných $\alpha_t(i)$ a $\beta_t(i)$ je možné určiť aj najpravdepodobnejší stav, v ktorom sa proces nachádza v čase t. Pre tento účel zadefinujme premennú $\gamma_t(i)$ ako pravdepodobnosť, že proces je v čase t v stave q_i

$$\begin{aligned} \gamma_t(i) &= P(q(t) = q_i) \\ &= \frac{\alpha_t(i)\,\beta_t(i)}{P(O \mid \lambda)} \\ &= \frac{\alpha_t(i)\,\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\,\beta_t(i)}. \end{aligned} \tag{1.28}$$

Normalizačný faktor $P(O \mid \lambda)$ váhuje $\gamma_t(i)$ tak, aby spĺňala podmienku

$$\sum_{i=1}^{N} \gamma_t(i) = 1 \tag{1.29}$$

Za pomoci $\gamma_t(i)$ môžeme určiť individuálne najviac pravdepodobný stav q_{i_t} v čase t, kde

$$i_t = \underset{i}{\operatorname{argmax}}[\gamma_t(i)], \quad 1 \leq i \leq N. \tag{1.30}$$

Aj keď predchádzajúci vzťah maximalizuje počet očakávaných korektných stavov (výberom najpravdepodobnejšieho stavu pre každý čas t), môžu nastať problémy s výslednou postupnosťou stavov. Napríklad, ak má model prechod do stavu, ktorý má nulovú pravdepodobnosť ($n_{ij} = 0$), potom takáto optimálna postupnosť stavov nemôže byť korektná. Uvedený postup jednoducho určí najviac pravdepodobné stavy, bez ohľadu na pravdepodobnosť výskytu týchto stavov.

Jedným z mnohých riešení tohto problému je modifikácia optimalizačného kritéria. Možností je viac, môžeme hľadať také postupnosti stavov, ktoré maximalizujú očakávané množstvo korektných dvojíc stavov (q_t, q_{t+1}) alebo trojíc stavov (q_t, q_{t+1}, q_{t+2}), atď. Tieto postupy sú dobré v niektorých typoch aplikácií, ale najviac používaným kritériom je nájdenie jedinej najlepšej postupnosti stavov. Inak povedané, maximalizujeme $P(Q \mid O, \lambda)$, čo je ekvivalentné maximalizácii $P(Q, O \mid \lambda)$. Postup na nájdenie jedinej najlepšej postupnosti stavov sa nazýva *Viterbiho algoritmus*[7], ktorý rieši problém rekurzívne a je založený na metódach dynamického programovania.

1.4.3 Trénovanie parametrov modelu

V tejto stati sa budeme zaoberať problémom, ako určiť parametre modelu $\lambda = (\mathcal{N}, \mathcal{M}, \pi)$ tak, aby sa maximalizovala pravdepodobnosť $P(O \mid \lambda)$ a aby model čo najlepšie popisoval danú postupnosť pozorovaní. Postupnosť pozorovaní, pomocou ktorej nastavujeme parametre modelu, sa nazýva *trénovacia postupnosť*.

Neexistuje známy analytický postup, ktorý by tento problém riešil. Môžeme zvoliť $\lambda = (\mathcal{A}, \mathcal{B}, \pi)$, ktorého $P(O \mid \lambda)$ je len lokálnym maximom, použitím iteračných postupov, ako je napríklad *Baum-Welchov algoritmus* alebo niektorú z gradientných techník.

Baum-Welchova metóda je iteračná, cyklicky opakuje odhady a aktualizuje parametre modelu. Definujme premennú $\xi_t(i, j)$, ktorá vyjadruje pravdepodobnosť, že daný model je v čase t v stave q_i a v čase $t + 1$ v stave q_j pri danej postupnosti pozorovaní O, nasledovne

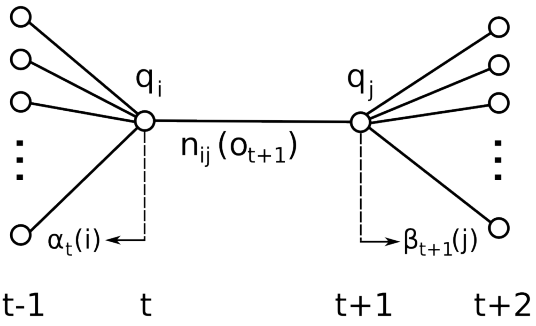
$$\xi_t(i, j) = P\big(q(t) = q_i, q(t + 1) = q_j \mid O, \lambda\big). \tag{1.31}$$

Z obrázku 1.4 vidieť, že túto pravdepodobnosť vieme vyjadriť vzťahom

$$\xi_t(i, j) = \frac{\alpha_t(i)\, n_{ij}\, m_j(o_{t+1})\, \beta_{t+1}(j)}{P(O \mid \lambda)} \tag{1.32}$$

pričom menovateľ $P(O \mid \lambda)$ je opäť použitý ako normalizačný faktor na váhovanie (aby platilo $0 \leq \xi_t(i, j) \leq 1$).

[7] Viterbiho algoritmus je popísaný v [1].



Obrázok 1.4: Ilustrácia výpočtu premennej $\xi_t(i,j)$.

V stati 1.4.2 sme definovali premennú $\gamma_t(i)$ ako pravdepodobnosť, že v čase t sme v stave q_i. Teraz ju môžeme vyjadriť prostredníctvom premennej $\xi_t(i,j)$

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{1.33}$$

Ak spočítame sumu $\gamma_t(i)$ cez všetky t, dostaneme číslo, ktoré by sme mohli interpretovať ako počet výskytov stavu q_i alebo očakávaný počet prechodov zo stavu q_i (ak zo sumy vylúčime posledný časový slot $t = T$). Podobne, sumu $\xi_t(i,j)$ cez $t = 1, \ldots, T-1$ možno interpretovať ako očakávaný počet prechodov zo stavu q_i do stavu q_j. Formálne zapísané:

$$\sum_{t=1}^{T-1} \gamma_t(i) \quad = \quad \text{očakávaný počet prechodov zo stavu } q_i \tag{1.34}$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) \quad = \quad \text{očakávaný počet prechodov zo stavu } q_i \text{ do stavu } q_j \tag{1.35}$$

Odvoďme teraz vzťahy potrebné pre metódu iteračného zlepšovania parametrov skrytého Markovovho modelu slova:

— π_i definujme ako očakávaný počet stavov q_i v čase $t = 1$

$$\hat{\pi}_i = \gamma_1(i); \tag{1.36}$$

— n_{ij} definujme ako pomer očakávaného počtu prechodov zo stavu q_i do stavu q_j, k očakávanému počtu prechodov modelu zo stavu q_i do ľubovoľného stavu

$$\hat{n}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}; \tag{1.37}$$

— $m_j(l)$ definujeme ako pomer očakávaného počtu časových okamžikov, koľkokrát je model v stave q_j a zároveň je pozorovaný výstupný symbol v_l, k očakávanému počtu časových okamžikov, koľkokrát je model v stave q_j

$$\hat{m}_j(l) = \frac{\sum_{t=1,o_t=v_l}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}; \tag{1.38}$$

kde $i = 1, \ldots, N$, $j = 1, \ldots, N$ a $l = 1, \ldots, L$. Ak definujeme aktuálny model ako $\lambda = (\mathcal{A}, \mathcal{B}, \pi)$ a použijeme ho pri výpočte vzťahov (1.36), (1.37) a (1.38) a zároveň si definujeme výsledok týchto výpočtov ako $\hat{\lambda} = (\hat{\mathcal{A}}, \hat{\mathcal{B}}, \hat{\pi})$, potom platia nasledovné vzťahy, ktoré dokázal Baum.

1. Počiatočný model λ definuje kritický bod pravdepodobnostnej funkcie, takže platí $\hat{\lambda} = \lambda$.

2. Model $\hat{\lambda}$ je lepší ako model λ v zmysle $P(O|\hat{\lambda})$, teda sme našli nový model, ktorý pri danej postupnosti pozorovaní má vyššiu pravdepodobnosť (lepšie modeluje túto postupnosť).

Na základe predchádzajúceho postupu, ak použijeme $\hat{\lambda}$ namiesto λ a opakujeme tento postup, môžeme zlepšovať pravdepodobnosť postupnosti O pri danom modele až po určitý limitný bod. Výsledok tohto procesu sa nazýva najviac pravdepodobný odhad parametrov skrytého Markovovho modelu. Treba podotknúť, že *forward-backward algorithm* vedie len k lokálnemu maximu a v mnohých prípadoch je týchto lokálnych maxím veľa. Vzťahy (1.36), (1.37) a (1.38) sa tiež dajú priamo odvodiť z tzv. *Baumovej prídavnej funkcie*

$$Q(\lambda, \hat{\lambda}) = \sum_{Q} P(Q \mid O, \lambda)\, log[P(O, Q \mid \hat{\lambda})], \tag{1.39}$$

ktorej riešenie vedie ku vzťahu

$$\max_{\hat{\lambda}} [Q(\lambda, \hat{\lambda})] \Rightarrow P(O \mid \hat{\lambda}) \geq P(O \mid \lambda) \tag{1.40}$$

Dôležitý aspekt tohto postupu je, že štatistické charakteristiky parametrov modelu (1.41) sa automaticky zachovávajú pri každej iterácii.

$$\sum_{i=1}^{N} \hat{\pi}_i = 1 \tag{1.41a}$$

$$\sum_{j=1}^{N} \hat{n}_{ij} = 1 \tag{1.41b}$$

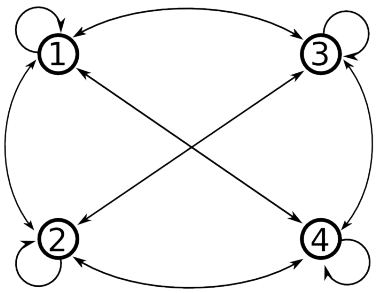
$$\sum_{k=1}^{N} \hat{m}_j(k) = 1 \tag{1.41c}$$

1.4.4 Typy skrytých Markovových modelov

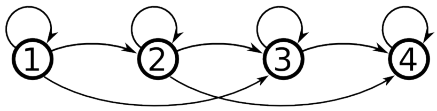
Doteraz sme uvažovali prípad, že v danom modele sa vieme v jednom kroku dostať z každého stavu do ľubovoľného iného stavu. Takýto model sa nazýva *plne prepojený* alebo *ergodický model*. Ergodický model definujeme ako model, v ktorom sa do ľubovoľného stavu vieme dostať z ktoréhokoľvek predchádzajúceho stavu na konečný počet krokov. Na obrázku 1.5 je príklad takého modelu. Každý koeficient n_{ij} modelu je väčší ako nula, jeho maticu $\mathcal{N}$ zobrazuje vzťah (1.42).

$$\mathcal{N} = \begin{bmatrix} n_{11} & n_{12} & n_{13} & n_{14} \\ n_{21} & n_{22} & n_{23} & n_{24} \\ n_{31} & n_{32} & n_{33} & n_{34} \\ n_{41} & n_{42} & n_{43} & n_{44} \end{bmatrix} \tag{1.42}$$

Pre niektoré aplikácie sa ukázalo, že iné typy skrytých Markovových modelov (v súvislosti s vlastnosťami signálov) lepšie modelujú dané signály ako ergonický model. Jeden takýto model je na obrázku 1.6. Tento model sa nazýva *Bakisov model* alebo model "zľava-doprava", pretože postupnosť stavov zviazaná s týmto modelom má vlastnosť, že s rastúcim časom rastie (alebo sa nemení) aj index stavu. Stavy sa menia zľava-doprava. Tento typ modelu má prirodzenú schopnosť modelovať signály, ktorých vlastnosti sa menia s časom.



Obrázok 1.5: Schéma ergodického modelu.



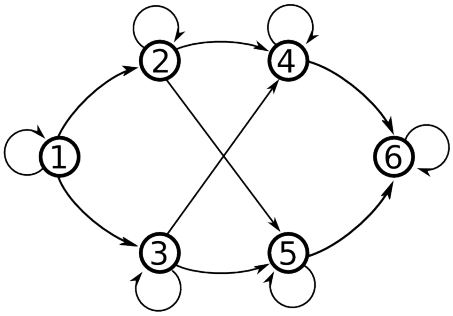
Obrázok 1.6: Bakisov model.

Takýmto signálom je aj rečový signál. Základnou vlastnosťou všetkých skrytých Markovovych modelov zľava-doprava je, že pre koeficienty prechodov do stavov platí

$$a_{ij} = 0, \quad j < i. \tag{1.43}$$

Teda nie je možné prejsť do stavu, ktorého poradové číslo je nižšie ako poradové číslo aktuálneho stavu. Súčasne pre počiatočnú pravdepodobnosť výskytu stavu platí

$$\pi_i = \begin{cases} 0, & i \neq 1; \\ 1, & i = 1; \end{cases} \tag{1.44}$$



Obrázok 1.7: Iný príklad modelu.

pretože postupnosť stavov musí začať v stave 1 a skončiť v stave N. Pri modeloch zľava-doprava sa často pridáva ďalšia podmienka, aby zmeny stavov neboli príliš veľké

$$n_{ij} = 0, \quad j > i + \delta \tag{1.45}$$

Pre model na obrázku 1.6 je δ rovné 2, teda v tomto modeli môže nastať zmena maximálne o 2 stavy. Matica $\mathcal{N}$ tohto modelu má nasledovný tvar

$$\mathcal{N} = \begin{bmatrix} n_{11} & n_{12} & n_{13} & 0 \\ 0 & n_{22} & n_{23} & n_{24} \\ 0 & 0 & n_{33} & n_{34} \\ 0 & 0 & 0 & n_{44} \end{bmatrix} \tag{1.46}$$

a súčasne je jasné, že posledný stav v modele zľava-doprava má vlastnosť

$$n_{NN} = 1, \tag{1.47}$$

$$n_{Ni} = 0, \quad i < N. \tag{1.48}$$

Hoci sme zatiaľ rozlíšili dva typy skrytých Markovových modelov, ergodický model a model zľava-doprava, existuje ešte veľa možností a variácií. Napr. na obrázku 1.7 je model, ktorý sa skladá z dvoch krížne spojených zľava-doprava modelov (všetky n_{ij} spĺňajú podmienku), ale pritom sa nejedná o striktný model zľava-doprava. Na tréningovú procedúru však nemajú vplyv žiadne podmienky

a obmedzenia v štruktúre modelu, pretože parameter, ktorý bol na začiatku nulový, bude nulový aj po tréningovej procedúre. Štruktúra modelu sa pri tréningu nemení.

Táto kapitola je veľmi stručným prehľadom niektorých princípov rozpoznávania reči, ktorú sme sa pre čitateľa snažili spracovať v rámci rozsahu tejto práce tak, aby výklad nebol príliš zložitý. Čitateľ v prípade záujmu môže siahnuť po publikáciách [1] alebo [3], z ktorých sme ostatne čerpali informácie pre potreby tejto kapitoly.

KAPITOLA DRUHÁ

IMPLEMENTÁCIA PROJEKTU

V dnešnej dobe rýchleho technického pokroku ľudia menia mnoho svojich dlhoročných zvykov. Knihy čoraz viac vytláča televízia, ľudia si namiesto listov posielajú emaily, voľný čas trávia čoraz viac v nákupných centrách a čoraz menej v prírode a nielen študenti si už zvykli na výhody počítača, ktorý im pomáha pri štúdiu alebo práci. Počítače, pred niekoľkými dekádami používané len úzkou skupinou ľudí a na konkrétne účely, dnes prenikajú do všetkých odvetví ľudského života a neustále rozširujú svoje pole pôsobnosti. Cieľom tejto práce je posunúť tieto hranice opäť o kúsok ďalej a preskúmať možnosti nasadenia počítača pri výučbe čítania.

Hlavnou časťou tejto diplomovej práce je implementácia programu, ktorý by mal slúžiť ako doplnkový nástroj pri výučbe čítania pre deti mladšieho školského veku. Program využíva techniky rozpoznávania reči ako prostriedok pre kontrolu správnej výslovnosti pri čítaní textu do mikrofónu. Okrem toho ponúka rôzne ďalšie možnosti ako sú prehrávanie slov či celých textov a nácvik výslovnosti slov alebo fráz.

V tejto kapitole budeme pojednávať o samotnej implementácii programu a problémoch s ňou spojenými. V oddiele 2.1 si prejdeme analýzu projektu, v oddiele 2.2 sa pozrieme na rôzne technológie a nástroje použité pri vývoji a implementácii, oddiel 2.4 je prehľad funkcionality, ktorú v sebe program zahŕňal v čase písania tejto práce a mohol by slúžiť aj ako užívateľská príručka a napokon v oddiele 2.5 uvedieme postrehy a poznámky ohľadom ďalšieho možného rozšírenia projektu z implementačnej stránky.

2.1 Analýza a návrh

V procese vývoja softvérového produktu rozoznávame niekoľko fáz, ktorých vhodne vybraná postupnosť má rozhodujúci vplyv na kvalitu vyvíjaného produktu a čas potrebný k jeho dokončeniu. V ďalšom jednotlivé fázy veľmi stručne popíšeme.

Vývoj softvérového produktu spravidla začína zberom požiadaviek. V prípade tohto projektu je "zákazník" a "vývojár" tá istá osoba, čo túto časť zjednodušuje. Po zbere požiadaviek nasleduje analýza, kedy sa zozbierané požiadavky analyzujú a vytvorí sa návrh budúceho produktu. Po dokončení návrhu sa môže začať s implementáciou samotného produktu, nemenej dôležitá je fáza testovania a ak sú všetky tieto fázy úspešne ukončené, môže sa pristúpiť k nasadeniu produktu do ostrej prevádzky.

Model vývoja softvérového produktu je popis, ako vo všeobecnosti jednotlivé fázy spolu súvisia a v akom poradí majú byť vykonávané. Taktiež popisuje vzťahy medzi nimi. Pre tento projekt sme použili tzv. *vodopádový model*, v ktorom jednotlivé fázy nasledujú po sebe jedna za druhou, pričom ďalšia fáza nezačne skôr, pokiaľ tá predchádzajúca nie je dokončená. Takto ideálne to však funguje málokedy a aj počas tohto projektu sme sa museli vrátiť od implementácie naspäť ku zberu požiadaviek, analyzovať dopad nových zmien na existujúci stav projektu, prepracovať návrh projektu a tieto zmeny potom implementovať.

2.1.1 Slovník pojmov

Každý projekt má svoje špecifické výrazy, ktoré zákonite vznikajú ako snaha minimalizovať objem textu nutného na popísanie problémov a udalostí týkajúcich sa projektu alebo skrátiť a zefektívniť vzájomnú komunikáciu osôb participujúcich na projekte. Z tohto dôvodu sa zavádza slovník pojmov[1], vysvetľujúci alebo definujúci často používané pojmy, ktorých význam nemusí byť intuitívny.

Dátový adresár – je adresár, v ktorom program hľadá konfiguračné súbory, dáta pre HTK, adresáre s príbehmi, grafiku atď.

[1] V praxi sa často používa anglický výraz *glossary*.

Diktát – funkcia programu, pri ktorej sa postupne prehrávajú jednotlivé slová za sebou s krátkou časovou prestávkou.
FP – funkčné požiadavky.
HTK – Hidden Markov Model Toolkit.
Neaktívny komponent – komponent, ktorý sa práve z nejakého dôvodu nedá použiť, na obrazovke sa zvyčajne prejavuje šedým výzorom. Angl. *disabled*.
NFP – nefunkčné požiadavky.
Otvorenie príbehu – načítanie textu príbehu na obrazovku.
Príbeh – text daného príbehu. V kontexte súborového systému ide o adresár, v ktorom sa nachádzajú všetky súbory týkajúce sa daného príbehu.
Slovo – pod pojmom slovo v programe rozumieme jedno slovo alebo krátku frázu (napr. slovné spojenie), príp. celú vetu v závislosti od toho, ako si užívateľ rozdelil text v `.xml` súbore.
Strana príbehu – každý dlhší príbeh je logicky rozdelený na niekoľko odstavcov. Program text celého príbehu zobrazuje po týchto odstavcoch – stranách, pričom v jednom momente je na obrazovke jedna strana textu.

2.1.2 Funkčné požiadavky

1. Program bude vedieť pracovať s adresármi obsahujúcimi súbory s textom príbehu, zvukovými a hlasovými vzorkami a ostatnými informáciami k príbehu (ďalej len *príbehmi*). Obsah adresára príbehu bude popísaný neskôr (v stati 2.3.3).

2. Program bude vedieť nájsť všetky adresáre obsahujúce relevantný príbeh (teda všetky príbehy), nachádzajúce sa v adresári na to určenom (dátový adresár).

3. Program ponúkne užívateľovi zoznam príbehov, ktoré sú k dispozícii na základe bodu č. 2-FP. Užívateľ si zo zoznamu vyberie príbeh, program mu zobrazí text daného príbehu na obrazovku a prehrá hlasovú ukážku nadpisu príbehu. Tejto akcii budeme hovoriť *otvorenie príbehu*.

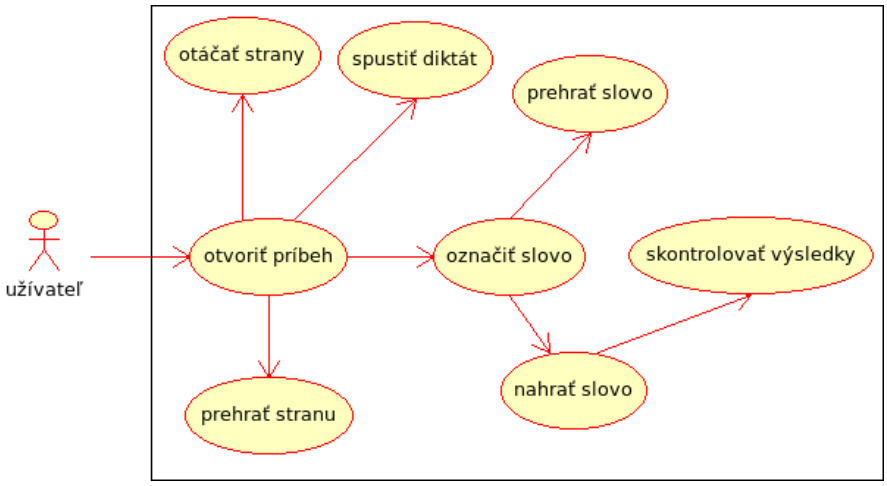
4. Po otvorení príbehu (bod č. 3-FP) program umožní užívateľovi prezerať jednotlivé strany príbehu.

5. Po otvorení príbehu (bod. č. 3-FP) môže užívateľ označiť niektoré zo zobrazených slov. Označenie slova je potrebné pre niektoré iné funkcie, ktoré pracujú s jednotlivými slovami a nie s celým zobrazeným textom.
6. Program umožní prehrať hlasovú ukážku aktuálne zobrazenej strany textu.
7. Funkcia *Diktát* prehrá hlasové vzorky jednotlivých slov, pričom medzi jednotlivé slová vloží pauzu ticha.
8. Počas vykonávania funkcie Diktát program podáva informáciu o tom, ktoré slovo je práve prehrávané (slová graficky označuje).
9. V prípade, že užívateľ pred spustením funkcie Diktát označil niektoré slovo na aktuálnej strane (bod č.5-FP), program začne diktovať od označeného slova (vrátane).
10. Vykonávanie funkcie Diktát bude možné kedykoľvek prerušiť.
11. Užívateľ si bude môcť prehrať hlasovú ukážku výslovnosti ľubovoľného označeného slova (bod č. 5-FP) zobrazeného na obrazovke.
12. Užívateľ bude môcť prečítať označené slovo (bod č. 5-FP) do mikrofónu, program vyhodnotí správnosť výslovnosti a výsledok mu zobrazí na obrazovku (v zmysle bodu č. 3-NFP).

Use case diagram zobrazujúci jednoduchý princíp používania programu na základe funkčných požiadaviek je zobrazený na obrázku 2.1.

2.1.3 Nefunkčné požiadavky

1. Program bude primárne implementovaný v prostredí operačného systému Linux.
2. Program je určený pre deti vo veku od sedem do deväť rokov, tzv. *mladší školský vek*.
3. V zmysle bodu č. 2-NFP užívateľské prostredie musí byť graficky prispôsobené danej vekovej kategórii, čo znamená, že:



Obrázok 2.1: Use case model.

- jednotlivé komponenty na ploche musia byť ľahko identifikovateľné a odlíšiteľné jeden od druhého;
- jednotlivé komponenty budú primárne popísané graficky (ikonou), avšak alternatívny textový popis musí byť tiež k dispozícii;
- výsledky rozpoznávania slova prečítaného do mikrofónu musia byť reprezentované graficky.

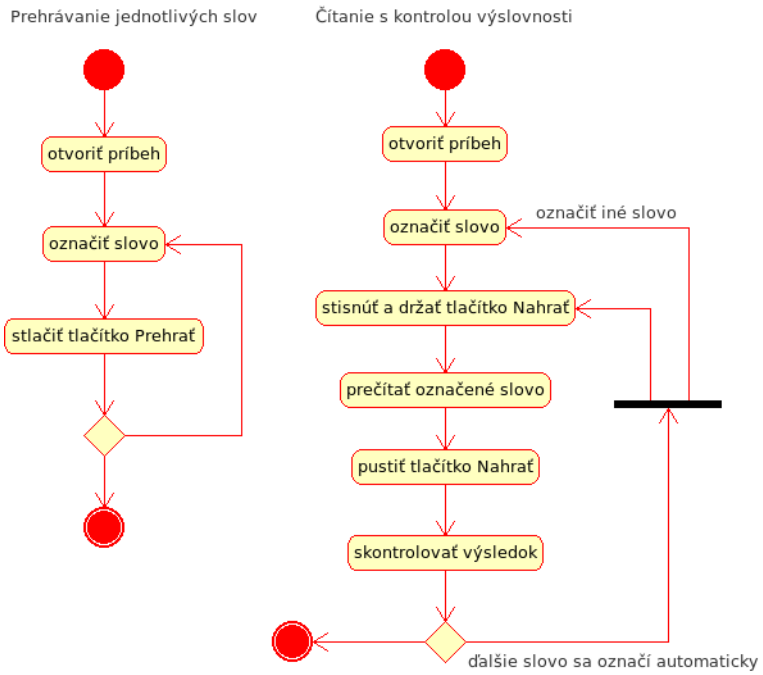
4. Program bude integrovaný do prostredia programu GCompris ako jedna z jeho aktivít.

5. Označené slovo v texte je graficky odlíšené od ostatných.

Obrázky 2.2 a 2.3 zobrazujú tzv. diagramy aktivít, ktoré popisujú návrh fungovania budúceho programu.

2.2 Použité nástroje a technológie

Výber správnych nástrojov je podstatnou súčasťou vývoja každého softvérového produktu. Poďme sa pozrieť na nástroje a technológie, ktoré sme používali pri implementácii tohto projektu.



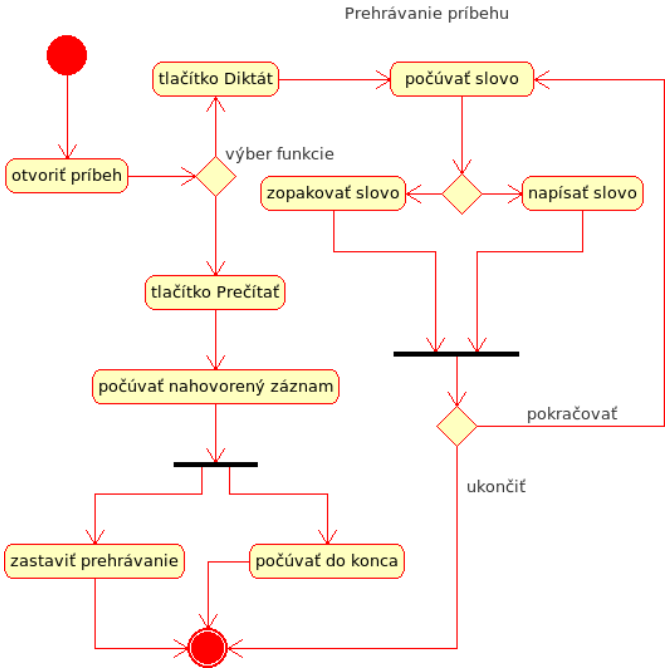
Obrázok 2.2: Activity diagram.

Linux je operačný systém založený na Unixe, ponúka stabilné prostredie a množstvo voľne dostupných nástrojov a programov, potrebných k vývoju softvéru. Počas realizácie tohto projektu bola použitá distribúcia Mandriva 2007.0.

C/C++ Samotný program je naprogramovaný v programovacom jazyku C++, pričom na mnohých miestach museli byť použité konštrukcie napísané v jazyku C. Spôsobuje to fakt, že všetky knižnice GNOME a GTK+ a aj program GCompris sú napísané v C, čo vylučuje použitie čisto iba C++.

GCompris je voľný edukačný softvér určený pre deti vo veku od dvoch do desiatich rokov. Program pozostáva z desiatok hier, nazývaných aktivity, rozdelených podľa účelu, cieľovej vekovej kategórie a obtiažnosti. Jednotlivé hry sú programy skompilované ako dynamické knižnice, ktoré si GCompris podľa potreby naťahuje a spúšťa.

GNOME canvas Projekt GNOME zastrešuje hneď niekoľko oblastí záujmu, pri práci na projekte sme však využívali knižnice pre vývoj softvéru, konkrétne GNOME a GTK+ a predovšetkým komponent GNOME canvas. GNOME can-



Obrázok 2.3: Activity diagram pre prehrávanie príbehov.

vas je grafická plocha s rozhraním na vysokej úrovni programovania[2] a vysokou výkonnosťou pre tvorbu štruktúrovanej grafiky [7]. GNOME canvas bol vybraný pre veľmi dobrú kompatibilitu s programom GCompris, nakoľko GCompris je na ňom založený.

HTK Toolkit je sada nástrojov, ktoré sú určené pre trénovanie skrytých Markovových modelov a na realizáciu rozpoznávania reči pomocou nich. V projekte sa HTK Toolkit využíva na kontrolu výslovnosti prečítaného slova do mikrofónu. Licenčné podmienky nepovoľujú ďalšiu distribúciu týchto nástrojov, preto si ich každý užívateľ bude musieť stiahnuť a nainštalovať sám.

Eclipse Vývojové prostredie Eclipse je zamerané predovšetkým na vývoj aplikácií v programovacom jazyku Java, ale s rozšírením *C/C++ Development Tooling Plug-in* sme ho využili na vývoj projektu v C/C++.

[2]Vysoká úroveň programovania znamená, že programátor sa pri jej používaní nemusí starať o základné problémy ako sú napr. prekreslovanie plochy pri zmene jej obsahu alebo generovanie udalostí jej jednotlivých objektov. O to všetko sa postará samotný GNOME canvas.

Typ	ASUS A6Km
Procesor	AMD Turion 64 (1600 MHz)
Pamäť	1GB DDR 333 SDRAM
Grafická karta	NVIDIA GeForce 7300 Go (256 MB)
Pevný disk	100 GB 5400 rpm
Operačný systém	Mandriva Linux 2007.0

Tabuľka 2.1: Špecifikácia notebooku.

XML Parser Pri implementácii projektu je použitá aj podpora XML, ktorá sa využíva na čítanie informácií z príbehových a konfiguračných súborov. Túto podporu zabezpečuje voľne dostupná knižnica `XML.h`, ktorú napísal Frank Vanden Berghen a ktorá je distribuovaná za podmienok licencie *BSD license*. [3]

Ďalej pri práci na projekte bol použitý program **Umbrello** na kreslenie diagramov v UML, program **Audacity** pre nahrávanie a úpravu hlasových ukážok, program **Inkscape** na vytváranie schématických obrázkov a **LaTeX** na vysádzanie textu tejto práce. Pre vývoj projektu bol použitý notebook, ktorého špecifikácia je uvedená v tabuľke 2.1.

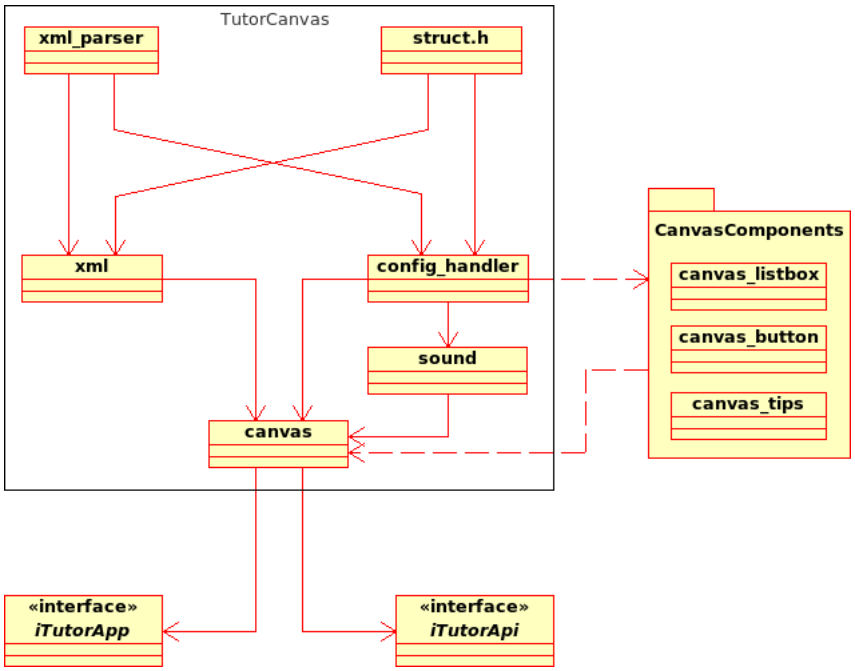
2.3 Implementácia

V ďalšom sa už pozrieme na konkrétnu realizáciu projektu, popíšeme rozdelenie celého programu na moduly a každý modul stručne popíšeme. Pod modulmi tu väčšinou rozumieme triedy. Schématický diagram tried je znázornený na obrázku 2.4.

2.3.1 Základné rozdelenie programu

Program, ktorého pracovný názov pre potreby tejto práce je *ReadTutor*, sa skladá z troch častí. Jadro samotného programu *TutorCanvas* a dve jeho rozhrania:

[3] Celé znenie licenčných podmienok je priložené v zdrojových kódoch zmieňovanej knižnice v súbore `xml_parser.h`.



Obrázok 2.4: Diagram tried.

TutorApp a *TutorApi*. ReadTutor môže byť skompilovaný a spustený dvoma spôsobmi. V prvom prípade je TutorCanvas skompilovaný ako dynamická knižnica[4], ktorú využíva rozhranie TutorApp. V tomto prípade je program spustený vo svojom okne ako obyčajná aplikácia[5]. V prípade druhom sa TutorCanvas skompiluje spolu s TutorApi a výsledkom je dynamická knižnica, ktorú využíva program GCompris ako jednu so svojich aktivít. Dôvody pre toto členenie programu sú dva; prvým dôvodom je fakt, že koncepcia integrácie projektu do GCompris prišla až počas implementácie pôvodnej verzie, ktorá bola navrhnutá ako samostatná aplikácia. Druhým dôvodom je omnoho jednoduchšie krokovanie[6] a rýchlejšie testovanie programu pri implementácii spoločných častí. Spôsob, ktorým sa ReadTutor skompiluje, závisí od predprocesorovej direktívy `GCOMPRIS_API`, ktorá sa nachádza v súbore `canvas.h` a ktorej použitie demonštruje obrázok 2.5.

[4]Pod pojmom dynamická knižnica máme na mysli tzv. *shared object* v Linuxe, resp. *dynamic link library* vo Windows.

[5]Takto skompilovaný program je v zdrojových kódoch nazývaný *Vanilla*.

[6]angl. *Debugging*.

```
#define GCOMPRIS_API true

#if GCOMPRIS_API
    /* kód pre GCompris */
#else
    /* kód pre "Vanilla" */
#endif
```

Obrázok 2.5: Ukážka použitia predprocesorovej direktívy.

2.3.2 Popis tried

Teraz si popíšeme jednotlivé triedy, z ktorých sa skladá TutorCanvas. Hlavičkový súbor `struct.h` obsahuje definície štruktúr a konštánt spoločných pre všetky ostatné triedy. Je k dispozícii vo všetkých ostatných hlavičkových súboroch s výnimkou `xml_parser.h`, ktorý sme už spomínali v oddiele 2.2.

Ďalšou triedou je trieda *Xml*. Jej úloha je pracovať s XML súbormi, obsahujúcimi označkovaný text príbehu a informácie o danom príbehu. Tieto dáta potom posúva vo vhodnom tvare ďalej do programu. K svojej činnosti využíva služby XML parsera.

Trieda *ConfigHandler* sa stará o čítanie a zapisovanie konfiguračného súboru `config.xml`. K svojej činnosti taktiež využíva XML parser. V hlavičkovom súbore tejto triedy `config_handler.h` sú definované predvolené nastavenia programu, tzv. *defaults*, ktoré sa pomocou metódy tejto triedy môžu exportovať do konfiguračného `.xml` súboru.

O prehrávanie a nahrávanie zvuku sa stará trieda *Sound*. Taktiež implementuje metódy pre prácu s fonetickými slovníkami a nástrojmi HTK. Podrobný popis práce tejto triedy s nástrojmi HTK ako aj celý proces nahrávania a vyhodnocovania hlasu budú popísané neskôr v tomto oddiele.

Trieda *Canvas* je hlavnou triedou celého programu a využíva služby všetkých už spomínaných tried. Jej úlohou je správa samotnej grafickej plochy GNOME canvas a komponentov na nej.

Tieto komponenty sú implementované ako nezávislé komponenty pre GNO-

ME canvas a ich použitie je veľmi podobné ako použitie GTK widgetov, konkrétne ide o komponenty *canvas tips*, *canvas button* a *canvas listbox*. Prvý spomínaný komponent je vlastne tzv. tooltip[7], malé okienko, ktoré sa objaví nad komponentom, ak nad ním prejdeme myšou a obsahujúce text alebo obrázok. Canvas button je obdoba bežného tlačítka, ktoré môže byť nakonfigurované na rôzne vzhľady a použitia. V konečnej verzii programu sa však používajú len tlačítka, ktorých vzhľad určuje obrázok alebo obrázková ikona. Na tlačítko môže byť nastavený tooltip. Tretí komponent dokáže zobraziť zoznam (napr. súborov, v prípade tohto projektu sa používa na zobrazenie dostupných príbehov).

2.3.3 Rozpoznávač reči

Ako sme už spomenuli, trieda *Sound* sa stará o obsluhu rozpoznávania reči. V tejto stati popíšeme algoritmus, ktorý sa na tento účel používa, načrtneme problémy, ktoré sme identifikovali a navrhneme riešenia, ako by sa tieto problémy dali vyriešiť.

Adresárová a súborová štruktúra dátového adresára

Začnime tým, že popíšeme obsah všetkých súborov, ktoré sa nachádzajú v dátovom adresári programu a rozoberieme ich obsah. Azda najdôležitejším súborom je `config.xml`. Obsahuje konfiguračné dáta, ktoré pri spustení ovplyvnia ako vzhľad, tak aj funkcionalitu programu. Nie všetky nastavenia sa dajú meniť z konfiguračného súboru, niektoré sú pevne zadané v súbore `config_handler.h`. Okrem súboru `config.xml` sa v dátovom adresári nachádzajú ešte ikona programu a súbor `AppWindow.glade`, ktorý popisuje užívateľské rozhranie programu TutorApp. V adresári `sh/` je niekoľko shell skriptov na rôzne hromadné konverzie audio súborov. Ku každému skriptu je aj jeho krátky popis. V adresári `img/` sú grafické súbory.

V adresári `_htk/` sú súbory spoločné pre všetky príbehy. `hmm.list` je súbor všetkých foném, `hrest.conf` je konfiguračný súbor pre HTK Toolkit,

[7] Po slovensky sa mu niekedy hovorí aj “bublinová nápoveda”, my však ostaneme pri anglickom “tooltip".

`hvite.sh` je shell skript, ktorý spúšťa program *HVite*. Program HVite je jeden z nástrojov HTK Toolkitu, ktorý realizuje samotné rozpoznávanie na základe dodaných modelov, fonetického slovníka, zvukového záznamu reči a `.lat` súboru. *HParse* je ďalší HTK program, slúži na spracovanie gramatiky, ktorá generuje rozpoznávané slová. Oba programy HVite a HParse, potrebné k fungovaniu programu ReadTutor, podliehajú licenčným podmienkam HTK Toolkitu, ktoré ich nedovoľujú distribuovať spolu s programom. Preto si užívateľ musí nastaviť cestu k nim v prípade, že sa nezhoduje s cestou nastavenou v `hvite.sh`. Posledným súborom v adresári `_htk/` je `all.hmm`, čo sú natrénované rečové modely. Tieto modely boli natrénované na príbehoch detí v rámci projektu *Multimediálna čítanka 2005* a testované na časti príbehov z *Multimediálnej čítanky 2006*, pričom úspešnosť rozpoznávania na neznámom rečníkovi bola 56% pri cca. 4000 slovách.

Ostatné adresáre, ktoré sa nachádzajú v dátovom adresári, by mali byť adresáre obsahujúce príbehy. Také adresáre by mali obsahovať súbor `index_seg.xml`, ktorý obsahuje okrem informácií o príbehu a autorovi aj text samotného príbehu v značkovacom jazyku XML. Súbor `phonet.txt` je fonetický slovník, obsahujúci slová, ktoré sú v danom príbehu použité. Sprievodná kresba príbehu je uložená v súbore `obrazok_pozadie.jpg` a napokon v podadresári `trimmed/` sú hlasové ukážky vo formáte `.wav`.

HTK Toolkit

HTK je sada nástrojov pre vytváranie skrytých Markovových modelov a predovšetkým pre vytváranie rozpoznávačov reči. Obsahuje trénovacie nástroje na odhadovanie parametrov modelov s využitím trénovacích hlasových vzoriek a im zodpovedajúcimi fonetickými prepismi a nástroje, pomocou ktorých rozpoznáva neznáme hlasové vzorky [8].

Bežné použitie nástrojov HTK je rozpoznanie neznámej hlasovej vzorky spomedzi konečnej množiny slov, ktorá je dopredu známa. HTK na základe natrénovaných parametrov modelu vyberie z množiny jedno slovo, ktoré sa najviac podobá danej neznámej vzorke. Tento princíp sa pokúsime využiť na rozpoznávanie správnosti výslovnosti.

Proces rozpoznávania

Ďalšia časť celého procesu začína v momente, kedy užívateľ klikne na nejaké slovo v príbehu, čím ho označí. Pri označení slova trieda *Sound* prejde fonetický slovník daného príbehu a vyberie z neho všetky fonetické prepisy označeného slova[8]. Z týchto prepisov potom trieda *Sound* vygeneruje všetky jeho prefixy a sufixy, ktoré vloží spolu s pôvodným slovom do nového fonetického slovníka. Do tohto slovníka ešte pridá všetky fonetické prepisy slov z pôvodného fonetického slovníka také, ktoré sú podobné s označeným slovom[9]. Z nového fonetického slovníka sa vygeneruje jednoduchá gramatika a z nej "lattice" súbor `words.lat`, ktorý ako jeden zo vstupných súborov pre HTK nástroje je reprezentáciou gramatiky množiny rozpoznávaných slov.

V tomto momente máme pripravené všetky súbory potrebné k rozpoznávaniu, okrem súboru obsahujúcemu hlasovú vzorku na porovnanie, ktorá sa vytvorí, keď užívateľ stlačí tlačítko pre nahrávanie. Vtedy sa začne nahrávanie vstupu z mikrofónu, ktoré sa ukončí v momente, kedy užívateľ tlačítko pre nahrávanie uvoľní. Zvukový záznam sa uloží do súboru `tmp.wav` v adresári príbehu a program hneď spustí skript `hvite.sh`, ktorý výsledok rozpoznávania uloží do súboru `tmp.rec` v adresári `_htk/`. Výsledkom je záznam z nového fonetického slovníka, ktorý bol vygenerovaný pri označení slova. Toto slovo je predané triede *Canvas*, ktorá výsledok vyhodnotí a graficky znázorní na obrazovke.

Algoritmus vzdialenosti textových reťazcov

Spomenuli sme pojem podobnosti slov. Budeme hovoriť, že dve slová sú si *podobné*, keď *vzdialenosť* ich textových reťazcov je malá vzhľadom na ich celkovú dĺžku. Táto podobnosť slov sa uvádza v percentách a vypočíta sa ako pomer vzdialenosti oboch reťazcov a dĺžky vzorového reťazca. Podobnosť reťazcov s_1 a s_2 môžeme zapísať ako

$$\mathcal{P} = \frac{\mathcal{D}(s_1, s_2)}{|s_1|},$$

[8]Jedno slovo môže mať viac fonetických prepisov v prípade, že sa pri hovorovej reči zvykne vyslovovať inak, ako by sa spisovne malo (napr. ľ).

[9]Pojem podobnosti dvoch slov zadefinujeme a vysvetlíme neskôr v tejto stati.

kde $|s_1|$ je dĺžka reťazca s_1 a $\mathcal{D}$ je vzdialenosť reťazcov s_1 a s_2.

Pre výpočet vzdialenosti textových reťazcov budeme používať tzv. *Levenshteinovu vzdialenosť*[10]. Táto vzdialenosť je definovaná ako počet "editačných" zmien, ktoré musíme vykonať, aby sme zo vzorového reťazca dostali cieľový. Tieto zmeny sú troch typov; zmena znaku, vloženie znaku alebo zmazanie znaku.

Dôležitým faktorom je aj celková dĺžka vzorového reťazca. Nech $\mathcal{D}$ je funkcia počítajúca Levenshteinovu vzdialenosť dvoch reťazcov. Majme reťazce s_1 a s_2, pričom nech dĺžka $|s_1| = 3$ a ďalej majme reťazce t_1 a t_2, kde $|t_1| = 8$. Nech $\mathcal{D}(s_1, s_2) = \mathcal{D}(t_1, t_2) = 3$. Potom reťazce t_1 a t_2 sú si navzájom podobnejšie ako reťazce s_1 a s_2, pretože pokiaľ potrebujeme 3 zmeny na transformáciu reťazca dĺžky 3, pravdepodobne dostaneme úplne iný reťazec, ktorý môže mať výrazne inú dĺžku alebo úplne rôzne znaky. Naproti tomu reťazec dĺžky 8 si po troch zmenách stále uchováva viac ako polovicu svojich charakteristických znakov.

Princíp Levenshteinovej vzdialenosti v projekte používame pri generovaní nového dočasného fonetického slovníka, kedy do neho vložíme aj tie slová, ktorých vzdialenosť od označeného slova je menej ako 50% vzhľadom na dĺžku označeného slova. Druhé použitie je pri vyhodnocovaní úspešnosti výslovnosti, kedy sa označené slovo porovná so slovom rozpoznaným a vypočíta sa ich vzájomná podobnosť.

Na základe tejto podobnosti sa označenému slovu zmení farba podľa preddefinovanej palety. Červené farby znamenajú zlú výslovnosť, oranžová znamená priemernú a zelená výbornú výslovnosť. Pri prechode myši ponad takto označené slovo sa výsledok zobrazí aj v tooltipe, ktorý obsahuje usmiatu alebo zamračenú tváričku, tzv. "smajlík".

Problémy a ich riešenia

Algoritmus, ktorý sme počas realizácie projektu vyvinuli a ktorý sme práve popísali, má niekoľko problémov, ktoré bude potrebné vyriešiť ešte predtým, ako sa projekt môže začleniť do edukačného procesu.

Najvážnejším problémom je, že program očakáva, že užívateľ do mikrofónu prečíta skutočne slovo, ktoré je označené. Pokiaľ užívateľ povie ľubovoľné

[10] angl. *Levenshtein string distance*.

iné slovo, program aj tak vypočíta najpravdepodobnejšie slovo zo svojho slovníka a toto vráti ako výsledok. Môže sa stať, že najvyššia pravdepodobnosť bude priradená práve označenému slovu, čo vyústi do situácie, kedy je toto náročky nesprávne slovo vyhodnotené ako úplne správne vyslovené. Tento problém spôsobuje predovšetkým nedostatočný počet slov vo fonetickom slovníku, nakoľko tento obsahuje len slová použité v danom príbehu. Jednoduchým riešením tohto problému by bolo používať rozsiahlejšie fonetické slovníky, obsahujúce napr. všetky slová z daného jazyka. Je však otázne, či pri použití slovníka, ktorý môže teoreticky obsahovať desiatky miliónov záznamov[11], bude proces rozpoznávania dostatočne rýchly a plynulý, ako je tomu teraz.

Pri testovaní sa ukázalo, že výsledok rozpoznávania môže byť vyhodnotený ako veľmi úspešný aj napriek tomu, že žiak sa zasekol, ale nakoniec slovo vyslovil správne. Zdá sa, že nástroje HTK si v hlasovej vzorke potom nájdu dané slovo aj napriek rušivým elementom, ktoré správne vyslovenému slovu predchádzali. Na potvrdenie alebo vyvrátenie tohto pozorovania bude potrebné vykonať rozsiahlejšie cielené experimenty. Tento jav však nastáva len zriedkakedy a len pri niektorých konkrétnych rečníkoch, spravidla tých, pre ktorých hlasy pracuje rozpoznávač so všeobecne vysokou úspešnosťou. Vo väčšine prípadov, pokiaľ sa rečník zasekol, slovo bolo vyhodnotené správne (teda červenou alebo oranžovou farbou).

Veľmi vážnym problémom je taktiež hlučné prostredie, čo sa veľmi jasne prejavilo počas experimentu. Veľa osôb v jednej miestnosti vyprodukuje už len svojou prítomnosťou množstvo rôzneho šumu a jemného hluku, ktorý znižuje kvalitu rozpoznávania. Riešením by mohlo byť aplikovanie protišumového filtra na testovanú hlasovú vzorku predtým, než vstúpi do procesu rozpoznávania. Softvérové produkty zamerané na prácu so zvukom (napr. Audacity) tieto funkcie ovládajú. Stačí pozbierať dostatočné množstvo vzoriek šumu, ktorý je charakteristický pre prostredie školskej triedy a tieto filtre z nich "natrénovať" podobne, ako sa trénujú Markovove modely. Pre lepšiu flexibilitu do úvahy prichádza vytvorenie viacerých druhov filtrov špeciálne pre rôzne prostredia.

Posledný problém, ktorý tu uvedieme a ktorý sa dosť jasne prejavil počas experimentu, je voľba spôsobu, akým sa bude spúšťať nahrávanie a rozpoznáva-

[11] Všetky slová vo všetkých svojich možných tvaroch a pádoch.

nie reči. Bola implementovaná filozofia "push-to-talk", kedy sa nahrávanie začne pri stlačení tlačítka a zastaví pri jeho uvoľnení. Tento spôsob sa ukázal ako príliš komplikovaný pre niektorých žiakov, zatiaľ čo pre niektorých iných znamenal prirodzený a jednoduchý spôsob, ako program efektívne využívať. Z tohto dôvodu by bolo vhodné pouvažovať nad implementáciou alternatívnych spôsobov pre nahrávanie hlasu. Jedno také riešenie je ostať pri jednoduchom kliknutí na tlačítko (jedno kliknutie na tlačítko spustí a druhé kliknutie zastaví nahrávanie) alebo po kliknutí na tlačítko nahrávať konštantný čas, príp. nahrávať dovtedy, pokiaľ sa na vstupe nevyskytne dlhšia pauza ticha. Tieto jednotlivé prístupy môžu byť implementované zároveň a použitie konkrétneho spôsobu môže závisieť od osobných nastavení daného užívateľa cez konfiguráciu programu.

2.4 Manuál programu

Doteraz sme v tejto kapitole písali o implementácii programu ReadTutor, ale zatiaľ sme si ho nepredstavili. V tomto oddiele popíšeme význam jednotlivých komponentov na obrazovke, vysvetlíme si ako program skompilovať a nainštalovať a na záver uvedieme návod, ako do programu pridať nové príbehy alebo modifikovať tie existujúce.

Ovládanie

Program ReadTutor sa ovláda výlučne myšou. Myšou sa kliká na tlačítka alebo na slová zobrazeného textu. Na ploche programu sú tri hlavné časti; 5 tlačítok v ľavom hornom rohu, vľavo dole je zobrazený sprievodný obrázok príbehu a informácia o autorovi príbehu a napokon vpravo je zobrazená otvorená kniha, do ktorej sa zobrazuje text otvoreného príbehu. Obrázok 2.6 znázorňuje plochu programu ReadTutor. Teraz popíšeme všetky tlačítka a vysvetlíme ich funkcie.

 Tlačítko *Otvorit'*, na ktorom je vyobrazený otvorený dokument, užívateľovi ponúkne zoznam všetkých dostupných príbehov. Kliknutie na názov príbehu otvorí daný príbeh, zobrazí text do knižky a prehrá hlasovú ukážku jeho názvu. Otvorenie príbehu sa dá zrušiť tlačítkom  . Na obrázku 2.7 je ukážka zoznamu príbehov.

Obrázok 2.6: Plocha programu ReadTutor.

V prípade, že užívateľ označil nejaké slovo v texte, tlačítko *Prehrať* je aktívne a kliknutím na neho sa dá prehrať ukážka daného slova. Vo verzii "Vanilla" sa dá prehrávanie kedykoľvek zastaviť kliknutím na tlačítko , ktoré sa pri prehrávaní objaví namiesto pôvodného tlačítka.

Tlačítkom *Nahrať* sa spustí nahrávanie slova. Tlačítko treba stlačiť a držať stlačené, vtedy sa nahráva. Pri uvoľnení tlačítka sa nahrávanie zastaví a spustí sa rozpoznávač. Po ukončení jeho činnosti sa označené slovo vyfarbí podľa výsledku rozpoznávača a označí sa ďalšie slovo v príbehu. Toto tlačítko je aktívne iba v prípade, že je označené niektoré zo slov príbehu.

Funkcia *Prečítať* spustí prehrávanie hlasovej ukážky celého textu, práve zobrazeného na obrazovke. Toto prehrávanie je možné zastaviť vo verzii "Vanilla", nie však v GCompris, kde to nie je možné, pretože GCompris na svojom rozhraní na to neposkytuje potrebnú funkcionalitu.

Obrázok 2.7: Ilustrácia otvárania príbehov.

Diktát je podobná funkcia ako *Prečítať* s tým rozdielom, že text nečíta ako celok, ale postupne diktuje jednotlivé slová, pričom medzi nimi necháva krátke medzery, počas ktorých môže užívateľ práve prečítané slovo zopakovať alebo zapísať. Pokiaľ je pri spustení tejto funkcie označené niektoré slovo v zobrazenom texte, diktát začne od toho slova, v opačnom prípade začne od začiatku strany. Prečítané slová sú postupne farebne odlišované a diktovanie je možné kedykoľvek zastaviť aj v GCompris.

Označené slovo je zvýraznené modrou farbou[12]. Slovo, ktoré bolo prečítané do mikrofónu pomocou funkcie *Nahrať*, je tiež farebne odlíšené od ostatného textu. Vpravo dole je farebná paleta, podľa ktorej sa slová farbia a v tooltipe nad slovom je zobrazený jeden z piatich "smajlíkov"; jeden z nich je zobrazený na obrázku 2.6. V rohoch obrázku otvorenej knižky sú šípky a , ktorými sa dá listovať v knihe.

[12]Farby označenia slov sa dajú nakonfigurovať pomocou konfiguračného súboru.

Inštalácia

V adresári `/ReadTutor/`so zdrojovými kódmi sú dve verzie zdrojovým kódov. Verzia v `/ReadTutor/eclipse/` sú zdrojové kódy uložené vo formáte pre vývojové prostredie *Eclipse*, ktoré stačí importovať. O správne skompilovanie sa už potom postará Eclipse. V prípade problémov je potrebné skontrolovať vlastnosti jednotlivých projektov, či sú správne nastavené cesty k hlavičkovým súborom a knižniciam potrebným ku kompilácii. Pri kompilovaní projektu TutorApp je potrebné nastaviť predprocesorovú direktívu `GCOMPRIS_API` na `false`, kým pri kompilácii projektu TutorApi na `true`.

V adresári `/ReadTutor/src/` sú zdrojové kódy pripravené na skompilovanie pomocou priloženého Makefile. Pre skompilovanie a nainštalovanie oboch verzií je potrebné napísať do príkazového riadku:

```
$ make
# make install
```

Presné inštrukcie pre skompilovanie a nainštalovanie programu sú uvedené v súbore `INSTALL`, priloženom ku každej verzii programu.

Je vhodné pred inštaláciou odinštalovať predošlú verziu (ak je nejaká nainštalovaná) príkazom

```
# make uninstall
```

a stiahnuť si najnovšiu verziu programu zo stránky projektu.

Nové príbehy

Budúci užívatelia programu ReadTutor si určite budú chcieť pridať nové príbehy, preto je vhodné uviesť, ako na to. V stati 2.3.3 sme popísali súbory, ktoré sa nachádzajú v adresári príbehu. Podrobný návod pre pridanie nového príbehu je popísaný v súbore `/doc/README` na priloženom CD.

2.5 Záverečné poznámky k implementácii

Nástroje HTK sme vybrali ako kvalitný prostriedok na realizáciu rozpoznávania reči v projekte a pokúsili sme sa ich použiť spôsobom, ktorý by vyhovoval našim zámerom. Na záver však musíme konštatovať, že napriek tomu, že HTK Toolkit je dobrý v tom, na čo je určený, teda na určenie slova z množiny slov s najvyššou pravdepodobnosťou, tento prístup sa ukazuje ako nedostačujúci resp. nevhodný v prípade, kedy je na výber len jedno slovo a my potrebujeme určiť, nakoľko správne bolo vyslovené. Riešenie tejto otázky bude vyžadovať sofistikovanejší prístup, prípadne jej vyriešenie na teoretickej úrovni.

Napriek tomu má tento spôsob aj svoje kladné stránky; je pomerne jednoduchý na implementáciu a prispôsobenie rôznym ďalším podmienkam, je nezávislý od jazyka, pre vytvorenie inej jazykovej lokalizácie stačí natrénovať nové Markovove modely pre daný jazyk a použiť text a hlasové ukážky v danom jazyku.

Ďalšou výhodou je jednoduchý spôsob, ako sa dá rozšíriť rozpoznávanie výslovnosti slov na rozpoznávanie väčších celkov, ako sú spojenia slov alebo celé vety. Toto rozšírenie nevyžaduje žiadne zmeny do programu, dokonca ani do natrénovaných modelov[13], stačí upraviť fonetický slovník a štruktúru delenia textu príbehov. Čítanie izolovaných slov je dobré pre deti, ktoré sa len učia abecedu a čítať ešte nevedia. Čítanie a opravovanie výslovnosti po spojeniach slov alebo po vetách je prirodzenejší spôsob čítania, pretože má bližšie k bežnej hovorovej reči a teda aj jeho prínos je väčší.

[13]Napriek tomu doporučujeme tieto modely natrénovať na nových vzorkách.

KAPITOLA TRETIA

VALIDÁCIA PROJEKTU

Ľudia robia chyby. Či už pri písaní dlhšieho textu, napr. knihy, tak aj pri písaní zdrojových kódov rozsiahlejšieho programu. Naše vlastné chyby sú pre nás často neviditeľné. Môžeme si sami po sebe viackrát skontrolovať náš vlastný text, nejaké chyby nájsť a úspešne ich opraviť, ale môže sa tiež stať, že niektoré chyby si nevšimneme a to aj napriek tomu, že vieme, ako to napísať správne. Písanie knihy a programu je svojím spôsobom podobné. Tak ako dobrý spisovateľ dá svoje dielo prečítať niektorým svojim budúcim čitateľom predtým, ako kniha vyjde, tak dobrý programátor by mal dať svoj program otestovať nejakej podmnožine budúcich užívateľov, pre ktorých je program určený.

Písanie programu je náročnejšie na chyby ako písanie knihy. Jedna drobná chyba v knihe, možno len nejaký preklep alebo nesprávne vyskloňovaný tvar slovesa, neznehodnotí knihu tak, ako nejaká malá chyba v programe. Ľudský čitateľ, ak si chybu vôbec všimne, si ju väčšinou dokáže opraviť sám pre seba a kniha v týchto prípadoch nestráca nič zo svojho obsahu. Počítač to sám nedokáže. Syntaktické chyby odhalí kompilátor, ale tie sémantické[1] si musí opraviť sám programátor.

Cieľom tejto práce je okrem iného preskúmať možnosti nasadenia programu, popísaného v predchádzajúcej kapitole, do pedagogickej praxe ako didaktickej pomôcky pri výučbe čítania. Našou snahou je rozšíriť množinu súčasne používaných didaktických pomôcok v edukačnom procese na elementárnom stupni školy. Program ReadTutor by mal vniesť do bežného vyučovania niečo nové, netradičné a každopádne veľmi zaujímavé pre žiaka i učiteľa. V tejto kapitole sa budeme

[1] Napr. možnosť prístupu do neinicializovanej pamäte, neupravenie všetkých častí programu pri implementácii novej požiadavky alebo použitie nesprávneho identifikátora.

zaoberať experimentálnym odskúšaním programu, popíšeme priebeh samotného experimentu a zhrnieme jeho výsledky.

3.1 Experiment na základnej škole

Osobitosťou experimentálnej metódy je skutočnosť, že sa za striktne dodržiavaných pravidiel usiluje odpovedať nielen na otázku, ako javy spolu súvisia, ale najmä na otázku, čo spôsobuje ich súvislosť, čo ich podmieňuje, čo je príčinou ich výskytu, ich povahy. Úsilie experimentálnej metódy potvrdiť či vyvrátiť teoreticky alebo empiricky zdôvodnené predpoklady, resp. hypotézy o príčinnonásledných súvislostiach, je dôležité nielen na budovanie teoretického systému pedagogiky, ale rovnako aj na praktické účely zefektívnenia edukačných procesov. Experiment sa vyznačuje syntézou teoretickej analýzy, zámernou manipuláciou (zmenou) nezávislých premenných a matematicko-štatistickou analýzou kauzálnych vzťahov.

Prístup pravého experimentu sa v teréne nedá často uplatniť. Nie je napr. možné náhodne vyberať žiakov z jednotlivých škôl a z jednotlivých tried a náhodne ich priraďovať experimentálnym a kontrolným podmienkam. Pre administratívne ťažkosti, časové i finančné obmedzenia, prípadne aj pre obávané reaktívne postoje sa preto často pracuje s celými triedami (tzv. intaktnými skupinami) a so školami, v ktorých sa vôbec dá výskum realizovať. Za takýchto podmienok má experiment povahu tzv. *kváziexperimentu*[2].

V našom prípade sme uskutočnili kváziexperiment[3]. Ten sa konal dňa 3. mája 2007 na Základnej škole Ostredková ul. č. 14 v Bratislave a zúčastnilo sa ho pätnásť žiakov a žiačok druhej triedy vo veku osem rokov; sedem chlapcov a osem dievčat. Cieľom experimentu bolo otestovať program v podmienkach, v ktorých by mal byť používaný. Pri tomto testovaní sme sa sústredili na:

- *Hľadanie chýb v programe* – pod pojmom "chyba" tu rozumieme hľadanie takých scenárov, ktorými by program mal vedieť prejsť a nie sú implementované alebo sú implementované nesprávne. Príkladom takej chyby je napr.

[2] [9] str. 173., kapitola *Experiment v edukačnom výskume*.

[3] V ďalšom však budeme pre jednoduchosť písať len *experiment*.

keď sa program neočakávane ukončí (spadne), nekonzistentnosť používateľského rozhrania (sú všetky tlačítka k dispozícii vtedy a len vtedy, kedy byť majú?), atď.

- *Ovládanie programu* – je ovládanie dostatočne intuitívne? Nie je príliš zložité? Dokáže žiak pracovať s programom tak, aby sa sústredil na samotnú podstatu štúdia a nemusel sa rozptyľovať nepohodlným ovládaním?

- *Užívateľské rozhranie* – skúmali sme, či je rozmiestnenie jednotlivých komponentov na obrazovke prirodzené a či deťom vyhovuje. Napríklad pri zadaní ľubovoľnej úlohy, kam približne sa žiak pozrie najprv alebo kam smeruje ukazovateľ myši, teda kde v prvom momente žiak hľadá komponent (napr. tlačítko), ktorým by mohol túto úlohu realizovať alebo informáciu, ktorú by pri realizácii mohol potrebovať.

- *Pochopenie programu* – pochopenie, ako program funguje a pochopenie zmyslu programu, na čo slúži, čo je jeho cieľom.

- *Fungovanie rozpoznávania reči* – v tomto prípade sme si všímali úspešnosť vyhodnocovania v porovnaní s tým, čo žiak skutočne povedal do mikrofónu.

Priebeh experimentu čitateľ nájde v oddiele 3.2 a jeho výsledky uvádzame v oddiele 3.3.

3.2 Priebeh experimentu

Experiment sa začal približne o štrnástej hodine, kedy som prišiel na Základnú školu Ostredková a stretol som sa s pani učiteľkou PaedDr. Luciou Sikovou, ktorá mi pomáhala experiment organizovať. Udalosť sa konala v školskom klube detí[4], nakoľko sme nechceli vstupovať do dopoludňajšieho vyučovacieho procesu.

Pre účely experimentu sme mali dohodnutú prázdnu miestnosť, pretože program je pri prevádzke citlivý na okolité zvuky. Žiaľ, plány sme museli zmeniť,

[4]Školským klubom detí sa kedysi hovorilo aj školská družina.

Obrázok 3.1: Žiak základnej školy skúša program ReadTutor.

takže experiment sa nakoniec odohrával v triede plnej žiakov. Počas družiny žiaci prichádzali zo záujmových krúžkov a odchádzali domov, čo malo za následok veľmi dynamicky sa meniace akustické prostredie ako aj premenlivý počet žiakov v triede. Vŕzganie dverami, hluk z chodby, šušťanie vody, šepkanie a tlmená reč, vŕzganie lavicami a stoličkami alebo dupot v triede či snaha o kolektívne testovanie mikrofónu, to sú niektoré z mnohých faktorov, ktoré vplývali na úspešnosť správneho fungovania programu.

Samotný experiment sa skladal z troch častí. Vysvetlenie a predstavenie programu, individuálne vyskúšanie programu žiakmi a diskusia[5]. Otázky sme sa žiakov pýtali prevažne individuálne, kým všeobecná diskusia prebiehala v prestávkach medzi skúšaním. Tieto tri časti sa medzi sebou prelínali, predovšetkým kvôli neustálej migrácii žiakov, takže celý proces sa nám tak rozdelil do niekoľkých iterácií, počas ktorých sme naraz pracovali s približne 4-5 žiakmi. Celkovo trval experiment približne dve a pol hodiny.

[5]Verbálna reflexia, viď. oddiel 3.3.

Obrázok 3.2: Žiaci so záujmom sledujú predvádzanie programu.

3.3 Výsledky experimentu

Po realizácii experimentu, resp. počas neho, máme niekoľko možností, ako z neho abstrahovať také informácie, aby sme z nich mohli vyvodiť relevantné výsledky. Prvou možnosťou je *spontánna reflexia*, kedy sledujeme reakcie žiaka na podnety počas experimentu. Môže ísť aj o prejavy verbálne, ale v tejto časti sa sústredíme hlavne na prejavy neverbálne.

Druhou možnosťou je *verbálna reflexia*. Verbálna reflexia môže byť súčasťou spontánnej reflexie, sú to komentáre žiaka počas experimentu, jeho vyjadrenia k experimentu počas neho alebo bezprostredne po ňom, ktoré žiak vyslovuje bez nášho podnetu, teda bez toho, aby sme sa ho explicitne pýtali. Môže sa jednať aj o konverzáciu medzi žiakmi, pri ktorej si vymieňajú názory k danej problematike, vtipkujú, atď., príp. ich konverzácia sa netýka danej témy, čo môže signalizovať nezáujem. Hlavnou časťou verbálnej reflexie sú však odpovede na naše otázky, priame či nepriame, pomocou ktorých sa snažíme zistiť názory žiakov.

Výtvarná reflexia je posledným spôsobom, ktorý tu uvádzame. Pri tomto spô-

sobe sa žiaci vyjadrujú výtvarne, čo môže byť pre deti v mladšom školskom veku jednoduchšie. Príkladom môže byť nakreslenie užívateľského rozhrania s pridaním vlastnej fantázie. Z viacerých kresieb sa potom dá určiť, čo vyhovuje väčšine žiakom alebo sa autor programu môže inšpirovať dobrým nápadom, na ktorý neprišiel sám jednoducho preto, lebo nie je dieťa v mladšom školskom veku.

Pri realizácii nášho experimentu a pri vyhodnocovaní jeho výsledkov vychádzame zo spontánnej a verbálnej reflexie, výtvarnú reflexiu neuvažujeme[6]. V ďalšom sa už budeme venovať samotným výsledkom experimentu.

3.3.1 Hľadanie chýb v programe

Testovanie programu odhalilo niekoľko drobných chýb vo vstupných dátach, z ktorých väčšina bola opravená priamo na mieste, konkrétne sa jednalo o nesprávne rozdelenie rozprávok na strany podľa zvukových nahrávok v .xml súboroch v prvom prípade a nesprávne pomenovaný súbor s hlasovou nahrávkou v prípade druhom. Taktiež sa našla aj jedna chyba v zdrojových kódoch; na okrajových stranách, teda na prvej resp. poslednej, sa po ukončení nahrávania hlasu do mikrofónu neoznačila šípka na predošlú resp. ďalšiu stranu ako "neaktívna". Toto miatlo žiakov, ktorí dočítali poslednú stranu a snažili sa obrátiť list.

3.3.2 Ovládanie programu

Skupinu žiakov, ktorí sa zúčastnili skúšania programu, by sme mohli rozdeliť do dvoch skupín. Do tej prvej by sme zaradili žiakov, ktorí pri počítači netrávia veľa času, ak vôbec nejaký a ktorí mali problém presne kliknúť myšou na niektoré z tlačítok. Pre týchto žiakov by bolo obtiažne zvládnuť akékoľvek jednoduché a užívateľsky príjemné ovládanie, preto ich nebudeme v tomto smere hodnotiť. V tomto prípade išlo o dve dievčatá a jedného chlapca. Pre ostatných bola filozofia tlačítkového rozhrania prijateľná a ovládanie programu zvládli rýchlo a bez problémov. Čo už však robilo problémy všetkým žiakom, bolo pochopenie fungovania tlačítka pre spustenie a zastavenie nahrávania, ktoré funguje spôsobom "Push-to-talk" (viac v implementačnej kapitole – oddiel 2.4). Z otázok, ktoré sme sa detí

[6]Z časových a administratívnych dôvodov.

pýtali, vyplynulo, že by im viac vyhovoval spôsob, kedy kliknutím na tlačítko spustia nahrávanie a ďalším kliknutím ho opäť zastavia, prípadne iný spôsob (iné spôsoby sú rozoberané v implementačnej kapitole – oddiel 2.5).

3.3.3 Užívateľské rozhranie

Pri pozorovaní reakcií žiakov na zadané úlohy sme si všimli opäť problém pri používaní nahrávania hlasu, ale v tomto prípade aj pri prehrávaní hlasových ukážok. Predpokladajme, že žiak sa rozhodol prečítať označené slovo do mikrofónu. Potom sa často stávalo, hlavne zo začiatku, že žiak hľadal prostriedok na spustenie prehrávania na samotnom slove alebo v jeho okolí. Prejavovalo sa to hlavne presúvaním kurzora myši nad označené slovo, prípadne v klikaní na slovo. Toto malo za následok zrušenie označenia daného slova a v prípade, že si žiak uvedomil svoj omyl, tak potencionálne kliknutie na neaktívne tlačítko[7]. Ak si svoj omyl neuvedomil, tak v niektorých prípadoch pokračoval v čítaní ďalších niekoľkých slov do mikrofónu, klikajúc na slová, pričom sa rozpoznávač ani nespustil.

Ďalší problém, ktorý sme počas skúšania programu zaznamenali, sa týkal rozmiestnenia ovládacích prvkov na ploche a najväčší problém bol opäť s nahrávaním. Všetky tlačítka sú umiestnené naľavo od textu. Na obrazovke to je niekoľko centimetrov v závislosti od nastaveného rozlíšenia. Užívateľ programu, snažiaci sa do mikrofónu prečítať označené slovo, sa tým dostáva do situácie, kedy sa musí sústrediť na dve miesta na obrazovke naraz. Toto síce nebol problém u väčšiny žiakov, ale o to väčšmi sa prejavoval práve u tej kategórie, ktorá s počítačom nemá veľa skúseností.

Z vyššie uvedeného sme prišli k záveru, že pre vyriešenie oboch problémov by bolo vhodné niektoré funkcie programu presunúť priamo na slová v texte, či už na rôzne tlačítka myši alebo prostredníctvom kontextového menu. Konečné slovo v tomto smere by však malo mať rozsiahlejšie testovanie programu v praxi na väčšej vzorke testovacích subjektov.

[7] Tlačítko už bolo v tom momente neaktívne, lebo nebolo označené žiadne slovo.

3.3.4 Pochopenie programu a celkový dojem

V stati 3.3.2 sme uviedli naše výsledky ohľadom pochopenia ovládania programu. Ale ako žiaci chápali zmysel programu? Pre nich to nebola didaktická pomôcka, ale zdroj zábavy. Viac ich zaujímalo, ako sa skončí aktuálna rozprávka alebo aké výsledky dosiahnu pri čítaní slov do mikrofónu, čomu sa napokon ani netreba čudovať.

Nás však viac zaujímalo, ako sa im program páčil a čo by na ňom vylepšili. Ako sme už uviedli pri výklade výtvarnej reflexie (strana 55), pre deti v tomto veku môže byť ťažké odpovedať na naše otázky podľa našich očakávaní, lepšie reagujú na otázky, na ktoré sa dá odpovedať iba áno alebo nie. Väčšina sa však zhodla, že sa im program páčil a niektorí si ho chceli vyskúšať aj druhý krát. Zaujímavá bola reakcia jednej žiačky, ktorá sa opýtala, prečo program neskúšame v miestnosti, kde je ticho, podľa nej by tam program fungoval lepšie.

Taktiež sme sa pýtali na formu, v akej by chceli dostávať výsledky rozpoznávania a kontroly výslovnosti. Všetci žiaci sa zhodli na tom, že filozofia usmiatych a zamračených tváričiek, tzv. “smajlíkov” a farebne odlíšených slov je dostatočne zrozumiteľná.

3.3.5 Fungovanie rozpoznávania reči

V oddiele 2.3.3 sme diskutovali o možnostiach zakomponovania rozpoznávania reči do projektu a o možnostiach jeho vylepšenia. Na tomto mieste nám ostáva konštatovať, že rozpoznávač reči fungoval na úrovni, na akej bol do projektu zaimplementovaný. Úspešnosť rozpoznania však kolísala v závislosti od aktuálnych akustických podmienok okolitého prostredia.

Uveďme niekoľko príkladov. Jedna žiačka pri čítaní dvoch strán textu dosiahla pozoruhodný výsledok, všetky slová okrem jedného program rozpoznal ako výborne vyslovené (tmavozelená farba – 95% a viac). Jej výslovnosť bola správna, ale rozhodne nie lepšia od výslovnosti predošlých žiakov. Dodajme, že počas čítania bola trieda takmer prázdna, v triede bola už len jedna žiačka a pani učiteľka a v triede bolo úplné ticho.

Iný žiak, naopak, po prečítaní jednej strany textu zistil, že všetky jeho slová sú rozpoznané ako červené alebo oranžové, teda slovami rozpoznávača veľmi zle vyslovené. Počas čítania tejto strany textu bol v triede čulý ruch, žiaci sa nahlas rozprávali, niektorí sa dokonca naháňali po triede. Ešte poznamenajme, že tento žiak tiež čítal slová pomerne správne. Po utíšení ruchu v triede sa aj jeho výsledky citeľne zlepšili, nie však na úroveň spomínanej žiačky.

Ďalšia žiačka čítala slová do mikrofónu a počítač ich vyhodnocoval s určitou úspešnosťou. Vtom sa počas čítania jedného slova vŕzgavo otvorili dvere, čo stačilo na to, aby rozpoznávač dané slovo vyhodnotil ako úplne zle vyslovené a to aj napriek tomu, že bolo vyslovené správne. Tá istá situácia sa opakovala pri zatvorení dverí. Rozpoznávanie ďalších slov potom prebiehalo normálne.

To nás vedie k niekoľkým záverom. Za prvé to podporuje náš predpoklad a síce, že rozpoznávač je veľmi citlivý na akýkoľvek hluk, ktorý zásadne vplýva na úspešnosť rozpoznania. Ďalším pozorovaním je fakt, že rozpoznávač nie je úplne nezávislý od rečníka; hlas niektorých osôb rozpoznáva lepšie ako hlas niektorých iných osôb. Pre prvý problém sme sa snažili poskytnúť riešenie v oddiele 2.3.3. Druhý problém by vyriešili kvalitnejšie natrénované rečové modely, natrénované predovšetkým väčším počtom rečníkov a na väčšom počte slov, čomu by sa v prípade nasadenia nástroja do edukačného procesu musela venovať zvýšená pozornosť.

3.4 Záverečné poznámky k experimentu

Na záver kapitoly sa zamyslíme nad prínosom projektu do pedagogickej praxe a predostrieme zopár návrhov do budúcnosti.

Výučba pomocou nášho softvérového nástroja je pre deti výbornou *motiváciou*; pre ne je to hra, pri ktorej sa učia a ani o tom nevedia. Rozvíja mnoho vlastností osobnosti žiaka, charakteristických pre výchovno-vzdelávací proces. Vedie deti k *samostatnosti*; zatiaľ čo jedno dieťa pracuje a ostatné ho pozorujú, tým sa vlastne pripravujú na úlohu, ktorú dostanú, keď na ne príde rad. Taktiež vedie k *tolerancii* v tých prípadoch, kedy nemôžu pracovať všetky deti naraz (napr. keď je k dispozícii málo počítačov) a musia čakať na svojich spolužiakov a tým

sa učia rešpektovať jeden druhého, posilňujú si interpersonálne vzťahy, osvojujú a utvrdzujú si široké spektrum prvkov prosociálnej výchovy. Rozvíja psychické procesy, ako napr. *vnímanie*; dieťa musí na obrazovke sledovať viac vecí súčasne[8] a *pozorovacie schopnosti*; deti samé pozorujú, či sa im prečítané slovo vyfarbilo dobrou farbou, sledujú text spolu s hlasovou ukážkou atď. Deti si do určitej miery rozvíjajú sebaregulačné vlastnosti, učia sa ohodnotiť samé seba (*sebareflexia*) a medzi sebou navzájom, čo rozvíja *sebahodnotenie* a *kritiku*. Pomáha *sebakontrole*; ovládať hnev alebo smútok z počiatočného neúspechu.

Budúcnosť projektu je v integrácii s programom GCompris. Ako jeho súčasť sa dostane do rúk ľudom na celom svete, čo poskytne viac príležitostí na jeho otestovanie a vylepšenie. Taktiež bude potrebné uskutočniť viac takých experimentov, aký sme uskutočnili v rámci tohto projektu. Tieto experimenty odhalia ďalšie slabé miesta programu a dopomôžu k jeho úspešnému vývoju.

Jedným z mnohých rozšírení projektu je aj možnosť archivácie výsledkov, ktoré boli dosiahnuté užívateľmi počas používania programu. Tieto výsledky by sa mohli uchovávať v centrálnej databáze, kde by napr. učiteľ mohol jednoducho kontrolovať pokrok svojich žiakov alebo odhaliť ich nedostatky v určitých oblastiach.

Počas experimentu sme u žiakov pozorovali záujem predovšetkým o rozpoznávač a prácu s ním. Žiaci nedostali žiadne konkrétne úlohy, čo majú s programom robiť a predsa väčšinu času strávili práve skúšaním a hrou s kontrolou výslovnosti. Tento výsledok nie je nijak prekvapivý, veď s prístrojmi na prehrávanie zvukov a hudby sa určite stretávajú na každom kroku, ale "rozprávať sa" s počítačom, to dnes ešte nie je bežné.

[8] A to aj v prípade, že zjednodušíme ovládanie v zmysle state 3.3.3.

Záver

V tejto práci sme sa snažili preskúmať možnosti využitia počítača pri výučbe čítania pre deti v mladšom školskom veku. Na splnenie nášho cieľa sme navrhli a následne implementovali program ReadTutor, ktorý sme vyskúšali pri praktickom experimente na základnej škole. Účelom tohto programu nie je nahradiť doterajšie metodiky, ale ich rozšíriť a obohatiť o ďalšiu alternatívu.

V úvode sme spomenuli tri pohľady, ktorými sa možno na prácu pozerať. Z pohľadu implementačného sme svoj cieľ splnili; implementovali sme program podľa našich predstáv, program, ktorý dokázal vzbudiť záujem u detí aj učiteľov. Tento program však nie je hotový, naše bádanie načrtlo mnoho možností jeho rozšírenia.

Preskúmali sme možnosti nástrojov HTK na realizáciu kontroly správnej výslovnosti, odhalili ich slabiny a poukázali na ich silné stránky. Taktiež sme navrhli možnosti ďalšieho postupu a riešenia problémov v tejto oblasti.

Prezentácia projektu na základnej škole, v prostredí jeho budúceho pôsobenia, priniesla výsledky aj v oblasti pedagogickej; pri experimente boli odhalené nedostatky programu a navrhnuté riešenia pre ich odstránenie. Pri predvádzaní sme mali možnosť pozorovať reakcie budúcich užívateľov a vypočuť si ich pripomienky, čo je zdrojom cennej inšpirácie. Zároveň sme identifikovali prínosy nášho projektu do pedagogickej praxe.

Budúcnosť projektu bude úzko spätá s edukačným softvérom GCompris. Pokiaľ sa podarí ReadTutor úspešne integrovať do GCompris a bude distribuovaný ako jeho súčasť, dostane sa do rúk ľudom na celom svete. Títo ľudia už budú reálni užívatelia s reálnymi potrebami a nápadmi a ich ohlasy a pripomienky pomôžu s napredovaním projektu.

Použitá literatúra

[1] PSUTKA, J. *Komunikace s počítačem mluvenou řečí*. Academia - nakladatelství Akademie věd České republiky, Praha, 1995.

[2] HOPCROFT, J. E., ULLMAN, J. D. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[3] NAGY, M. *Skryté Markovove modely a rozpoznávanie číslic*. Fakulta matematiky, fyziky a informatiky, Univerzita Komenského v Bratislave, 2004.

[4] Phonetic alphabet - wikipedia, the free encyclopedia. Internetová stránka. `http://en.wikipedia.org/wiki/Phonetic_alphabet`.

[5] RABINER, L. R., JUANG, B. H. An introduction on hidden markov models. *IEEE ASSP Mag. 3* (1986).

[6] RABINER, L. R. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE 3* (1989).

[7] Gnome developer documentation - reference manual. Internetová stránka. `http://developer.gnome.org/doc/`.

[8] YOUNG, S., EVERMANN, G., GALES, M., HAIN, T., KERSHAW, D., LIU, X., MOORE, G., ODELL, J., OLLASON, D., POVEY, D., VALTCHEV, V., WOODLAND, P. *HTK Book (for version 3.4)*. Cambridge University Engineering Department, December 2006.

[9] ŠVEC, Š. *Metodológia vied o výchove*. Iris, Bratislava, 1998.