Temat: “Transformacja tekstu pisanego na mowę”

Niniejszy zbiór podprogramów lingwistycznych, informatycznie zespolonych w moduł komplementarny, realizuje zadania związane z analizą tekstu pisanego, gdzie treść pisemnej wypowiedzi cyfrowej podlega ewaluacji (ocenie) syntaktycznej. Sukcesywnie zostaje ona przetransformowana w adekwatny zbiór dźwięków reprezentujących pierwotny zbiór treści pisanej. Napotkane struktury językowe podczas analizy, zapisane w bazie przydzielonej aktualnie rozpatrywanemu językowi, zostają poddawane transkrypcji. Język może zostać wykreowany na podstawie tekstu źródłowego i po operacjach diagnostycznych, samplowaniu, strukturyzacji jest używany przez stały algorytm TTS (”Text To Speech”). Możliwa jest również walidacja oraz korekta utworzonej bazy językowej.

Przegląd bibliografii

[D1] Praca magisterska autorstwa Roberta Rodakowskiego nakreślająca spektrum zagadnień i problemów związanych z konkatenacyjną syntezą mowy w ujęciu praktycznym. Opracowanie to, stanowi swego rodzaju analizę etapów prac, przez jakie należy „przejść” tworząc syntezator mowy. Problematyka tematu została ukazana w takim świetle, aby praca ta mogła stanowić punkt wyjścia w drodze do zgłębienia tajników metody konkatenacyjnej, oraz stworzeniu własnego syntezatora mowy.

[I1] Strona internetowa Wikipedii. Zacytowano z niej encyklopedyczne informacje na temat syntezy mowy.

[I2] Strona internetowa Wikipedii. Zaczerpnięte zostały z niej informacje dotyczące firmy ”IVO-Software”.

[I3] Strona internetowa twórcy syntezatora ”DANT”. Zawiera informacje o syntezie mowy, sposobach testowania.

[I4] Strona internetowa przedstawiająca ogólne informacje na temat algorytmów zamieniających tekst pisany na mowę.

[I5.] Witryna internetowa Departamentu Fonetyki i Lingwistyki Uniwersytetu Londyńskiego. Posiada wiele informacji lingwistycznych dotyczących mowy, które niezbędne są przy tworzeniu syntezatorów.

[A1] Artykuł autorstwa Adama Wojciechowskiego z Instytututu Informatyki Politechniki Łódzkiej. Przedstawia propozycję wykorzystania elementów technologii syntezy i rozpoznawania mowy do sterowania wirtualną galerią sztuki.

[K1] Książka pióra Kenta Reisdorpha. Przedstawia środowisko programistyczne ”Borland C+ + Builder”, w którym stworzono aplikacje na potrzeby niniejszej pracy.

[K2] Podręcznik do pracy w programie ”3D Studio Max” stał się przydatny w trakcie tworzenia grafiki.

[K3] Książka wyjaśniająca zależności fonetyczno – fonologiczne języka polskiego.

I. Analiza rozwiązań

I.1 Analiza istniejących rozwiązań

W tym punkcie zostanie przedstawiona analiza istniejących rozwiązań problemu syntezy mowy. Przybliżone zostaną także ogólne informacje dotyczące tego procesu, stosowane algorytmy syntezy oraz istniejące rozwiązania programowe.

”Proces syntezy mowy dzielimy na dwa etapy. W pierwszym z nich program wydobywa z wprowadzonej frazy jak największą ilość informacji lingwistycznych – stara się zrozumieć tekst. Etap ten nazywany jest przetwarzaniem języka naturalnego NLP (ang. Natural Language Processing). Później następuje utworzenie dźwiękowej wypowiedzi frazy na podstawie zdobytych o niej informacji – jest to etap cyfrowego przetwarzania sygnału DSP (ang. Digital Signal Processing). W obu wyżej wymienionych etapach jest wykonywanych wiele pośrednich kroków, wymagających sporej wiedzy lingwistycznej i matematycznej.”[1]

Pierwsze próby syntezy głosu, pochodzą z roku 1773. Badania w owym czasie prowadził Ch. G. Kratzenstein, który posiadał tytuł profesora fizjologii na uniwersytecie w Kopenhadze. Wiedza z anatomii oraz zainteresowanie instrumentami muzycznymi, doprowadziły do zbudowania niezwykłych organów, jakie generowały dźwięki samogłoskowe. Dokładny opis budowy tego urządzenia można znaleść w ”Mechanismus der menschlichen sprache nebst beschreibung einer sprechenden maschine” autorstwa Wolfganga von Kempelena (1791). Urządzenie doczekało się zrekonstruowania w roku 1835 i instalacji w Dublinie. Kolejne osiągnięcia należą do Josepha Fabra (1846-Londyn), który zbudował śpiewającą maszynę nazwaną ”Euphonia”, oraz R.R.Riesz (1937-USA), którego urządzenie wiernie odwzorowywało mowę. Homer Dudley, w czasach, gdy znana już była elektryczność zaprezentował urządzenie oparte właśnie na niej. Ochrzczono je imieniem ”Voder” i zaprezentowano w Nowym Jorku 1939 roku. Natomiast 1950 rok należał do Frank’a Cooper’a, urządzenie skanowało tekst wiązką światła a następnie go odczytywało. Od roku 1970 nastąpiło trwałe związanie syntezy mowy z komputerami. Obecnie prowadzi się wiele bardzo zaawansowanych prac nad udoskonaleniem syntezatorów, ich głosy nie są już bezosobowe. Można określać wiek, płeć a czasem i emocje w głosie, podczas odczytu tekstu przez maszynę.

[1] http://pl.wikipedia.org/wiki/Syntezator_mowy

Jedną ze starszych metod (początek epoki komputerów) jest formantowa synteza głosu. Polega ona na modulacji częstotliwością sygnału, tak by otrzymać jak najbardziej podobny dźwięk do formantów głosek. Otrzymywane efekty nie są jednak zadowalające.

Późniejszym algorytmem syntezy mowy, jest metoda artykulacyjna. Jej działanie oparte jest o budowę ludzkiego aparatu mowy. By wygenerować głoskę potrzebne jest około sześćdziesięciu parametrów. Ten rodzaj syntezy jest w zasadzie tylko teoretyczny i nie jest rozpowszechniony.

Zakładając, że uzyskany dźwięk ma być jak najbardziej zbliżony do ludzkiego głosu, najprostszym rozwiązaniem jest zastosowanie jego próbek (tak zwanych sampli, lub segmentów). *”Metoda konkatencyjna polega na nagraniu dużej bazy prawdziwego głosu lektora (tzw. Baza segmentów), jej oznaczeniu i przetworzeniu, a następnie w procesie syntezy mowy wybieraniu, modyfikowaniu oraz składaniu sygnału mowy z fragmentów wcześniejszych nagrań. Technika ta pozwala na uzyskanie mowy najbardziej naturalnej*.”[2]

Procesy zachodzące w poszczególnych etapach syntezy mowy zostały wyszczególnione poniżej.

W etapie NLP wyróżniamy następujące działania:

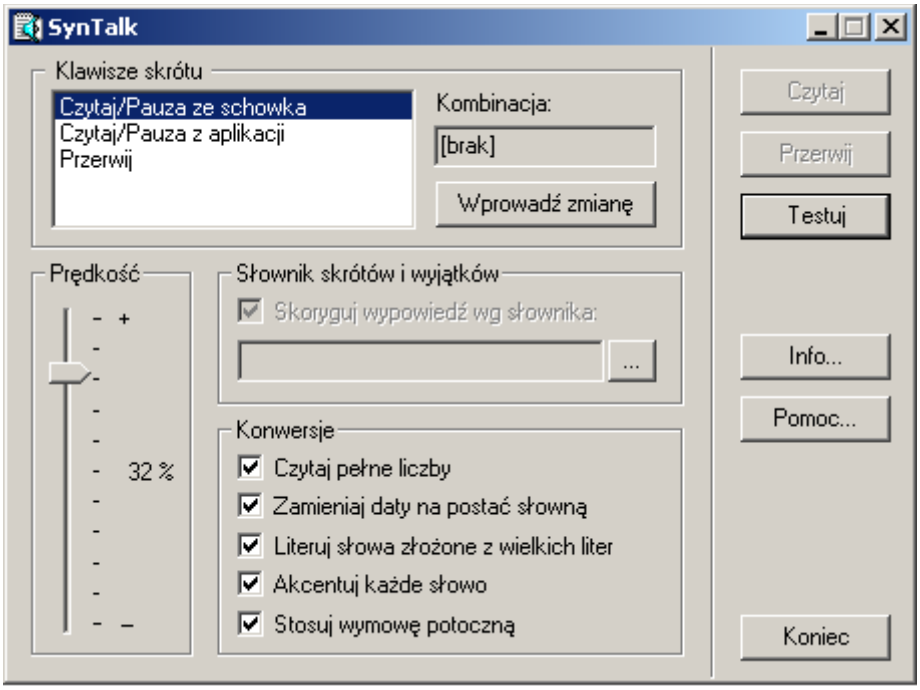
- analiza tekstu,
- transkrypcja fonetyczna,
- generowanie prozodii,
- wysłanie danych do DSP.

Natomiast w etapie DSP zachodzą takie procesy jak:

- odbiór fonemów i prozodii z etapu NLP,
- dekodowanie / dekompresja segmentów (dzięki bazie segmentów),
- dopasowanie prozodii,
- konkatenacja, czyli łączenie segmentów,
- synteza sygnału,
- mowa.

[2] Ibidem

Obecne rozwiązania programowe pozwalają wiernie odtwarzać mowę. Poniżej chronologicznie zostały opisane programy od najstarszych do najnowszych i zarazem najbardziej zaawansowanych.



Rys. 1.1.1 Program syntezy mowy SynTalk

Na Rys. 1.1.1 został przedstawiony program SyntTalk stworzony przez firmę Neurosoft (http://www.neurosoft.pl). Jest on jednym z najstarszych syntezatorów języka polskiego. Poza samym syntezatorem w pakiecie zostały zawarte takie programy jak:

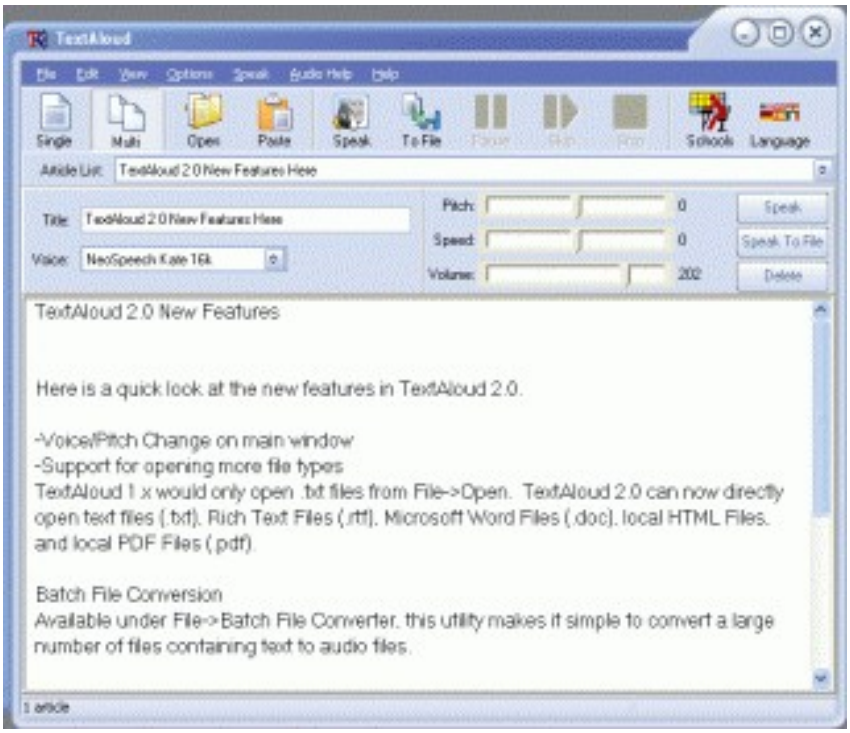
· OleTalk, serwer OLE dla obiektów typu ”Komentarz słowny”,

· EditTalk, program do odczytywania informacji wprowadzanych z klawiatury,

· ClockTalk, mówiący zegarek i terminarz,

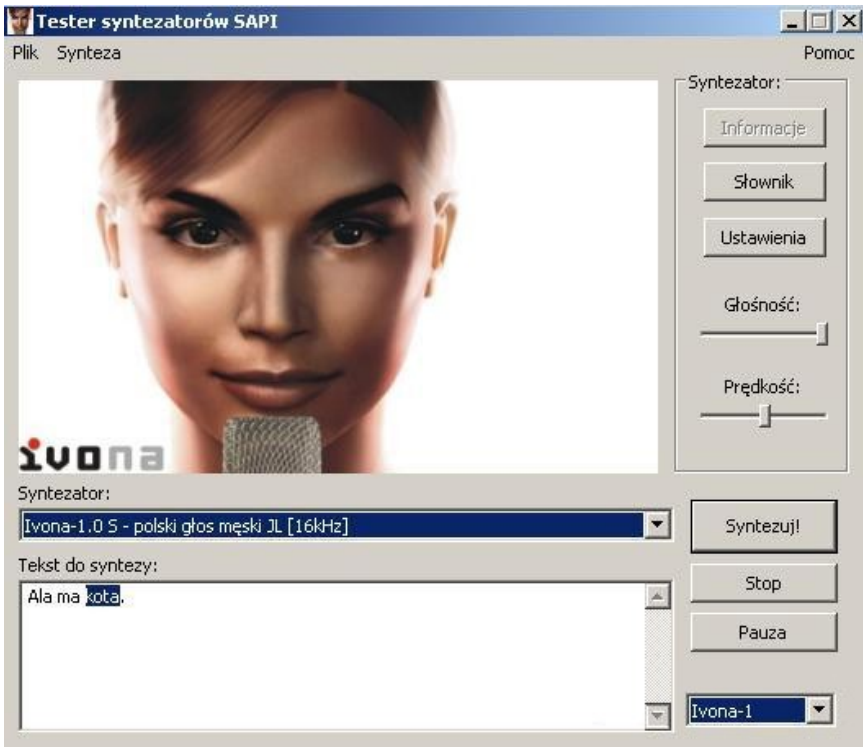
· NetTalk, nadajnik głosowych komunikatów w sieci LAN,

· TrueTalk, osobisty doradca.



Rys. 1.1.2 Program RealSpeech

Na Rys. 1.1.2 przedstawiono zrzut ekranowy programu RealSpeech. Aplikacja może być zaimplementowany w urządzeniach do nawigacji satelitarnej, call center. Strona producenta http://www.nuance.com/



Rys. 1.1.3 Syntezator IVONA

Program IVONA Rys. 1.1.3 jest jednym z najbardziej zaawansowanych syntezatorów na świecie. [i2] *”IVO Software Sp. Z o.o. – firma informatyczna założona w 2001 r. W Gdyni przez Michała Kaszczuka i Łukasza Osowskiego – absolwentów Politechniki Gdańskiej. Siedziba firmy mieści się na terenie Pomorskiego Parku Naukowo-Technologicznego w Gdyni – www.ppnt.pl (...) IVO Software jest producentem syntezatorów mowy z rodziny IVONA, który w roku 2006 i 2007 został uznany za jeden z najlepszych na świecie w prestiżowym konkursie Blizzard Challenge.”* [3]

I.2 Koncepcja rozwiązania

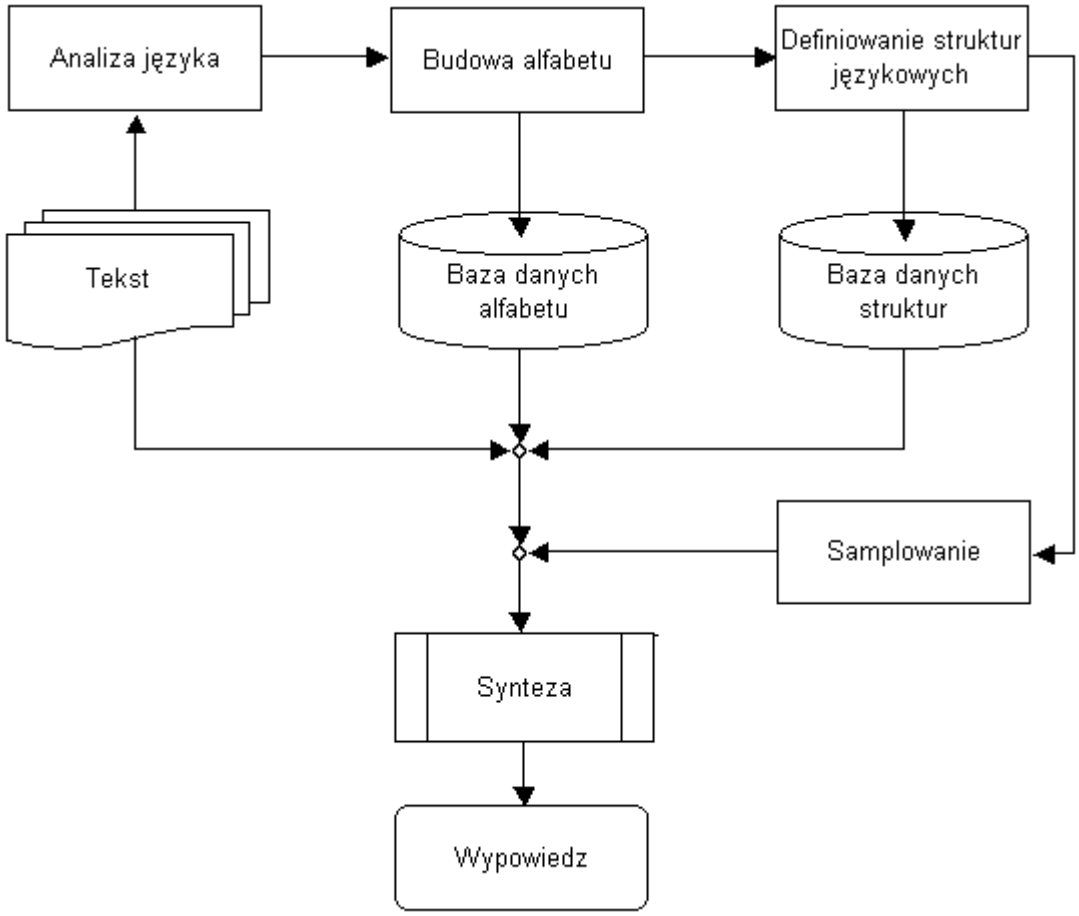
”W toku ewolucji, przez setki tysięcy lat, natura wytworzyła jeden z najdoskonalszych systemów służących do komunikacji – mowę. Istnieje wiele języków i dialektów, jednakże aparat mowy w każdym przypadku jest ten sam. Powstające za jego pomocą dźwięki niosą odbiorcy całe mnóstwo informacji. Oprócz tych związanych z treścią – tzw. semantycznych, są także osobnicze, pozwalające na rozpoznanie osoby mówiącej, oraz emocjonalne, pozwalające odgadnąć stan osoby mówiącej, jak na przykład jej zdenerwowanie, wzruszenie, rozbawienie, a także całe mnóstwo innych informacji takich jak status społeczny, wykształcenie, pochodzenie, a nawet stan zdrowia. Wszystko to jest możliwe za sprawą procesu artykulacji, czyli kształtowania dźwięków mowy ludzkiej, oraz percepcji. Natura doprowadziła je do perfekcji. Są one bardzo złożone, towarzyszą im także inne procesy, nie mniej skomplikowane, przez co angażują one ogromne zasoby mózgu ludzkiego. Co ciekawe, znakomita większość z nich zachodzi jednak bez udziału świadomości, są dla nas „automatyczne”, naturalne. Cała doskonałość tego systemu powoduje wrażenie, iż mowa, a nawet cały proces porozumiewania się, zdaje się być prosty, łatwy, wręcz banalny. Jednakże bardziej wnikliwe obserwacje tych zjawisk, a przede wszystkim próby odtworzenia i wykorzystania ich w świecie techniki, ukazują nam stopień komplikacji tego procesu.” [4]

Na podstawie wniosków, wyciągniętych w trakcie testowania własnego algorytmu o założeniach zbliżonych do algorytmu konkatencyjnego (wzbogaconego w analizę struktur językowych) została opracowana koncepcja rozwiązania. Istotną różnicą między

[3] http://pl.wikipedia.org/wiki/Ivo_Software

[4] Robert Rodakowski ”Opracowanie modułu konkatenacyjnej syntezy mowy polskiej na podstawie tekstu do zastosowań w komputerowej terapii wad wymowy dzieci” Akademia Górniczo – Hutnicza im. Stanisława Staszica w Krakowie.

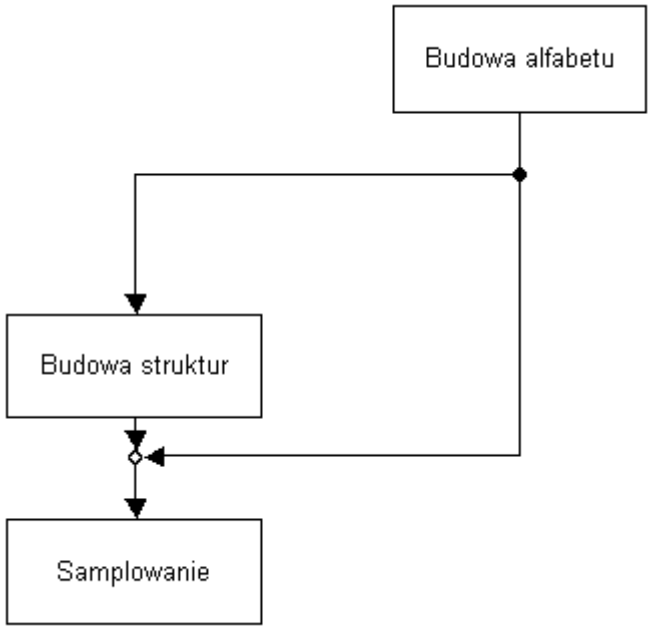
przedstawionym tu rozwiązaniem, a już istniejącymi jest liczba definiowalnych języków. Syntezatory mowy TTS dostępne na rynku mają wbudowaną strukturę budowy języka w kod programu – oznacza to, że nie można ingerować w raz zaprogramowane zasady odczytu tekstu. Przedstawiony w niniejszej pracy pakiet programów nazwanych ”SYNT” zawiera syntezator mowy (TTS). Posiada także aplikacje umożliwiające zdefiniowanie zasad odczytu tekstu w definiowanym języku, na podstawie przykładowego tekstu źródłowego i zapisanych struktur językowych. Jest to nowe rozwiązanie o znacznie większej złożoności od obsługi raz ustalonego języka w momencie kompilacji syntezatora.



Rys. 1.2.1 Schemat procesów transformacji języka z formy pisemnej na werbalną

Na Rys.1.2.1 został przedstawiony proces powstawania wypowiedzi werbalnej na podstawie tekstu pisanego. Podstawowymi procesami są; analiza języka oraz budowa alfabetu. Alfabet jest tworzony po wykonaniu analizy tekstu o odpowiedniej ilości znaków. Dłuższy tekst wykazuje lepsze wyniki analizy od tekstu krótkiego. Związane jest to z tym, że przy dłuższej kombinacji znaków występuje większe prawdopodobieństwo powstania powiązań językowych, oraz użycia symboli znakowych, niż w tekście krótkim. W wielu językach alfabet posiada znaczną ilość znaków oraz fonemów. Do ich całkowitego poznania konieczne

jest wprowadzenie większej ilości tekstu (minimum kilka stron). Jeśli mają być dodatkowo analizowane cechy fonologiczne relewantne (dźwięczność lub ubezdźwięcznienie) oraz generowanie prozodii (akcent, intonacja, iloczas), wtedy algorytm jest wysoce skomplikowaną analizą struktur.



Rys. 1.2.2 Relacja między modułami analizującymi budowę języka

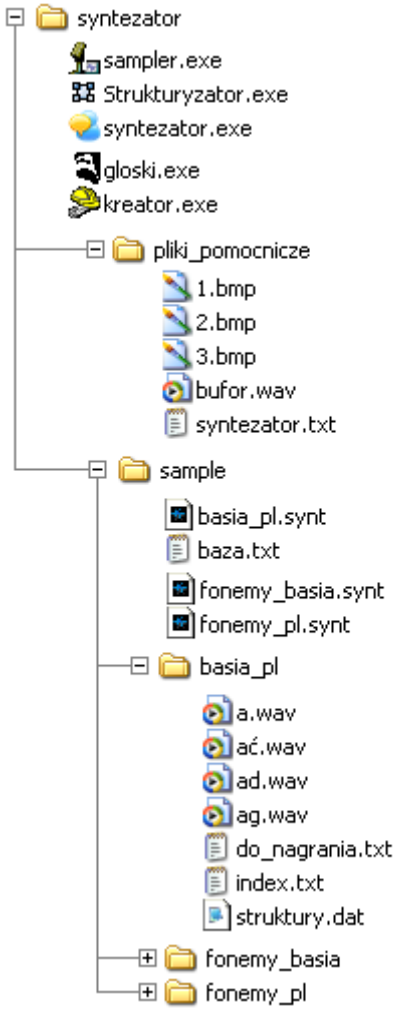
Proces budowy języka został podzielony na moduły, przypisane poszczególnym aplikacjom o wspólnej nazwie pakiet ”SYNT”. Wszystkie aplikacje połączone razem rozpoznają budowę języka, jaki jest używany przez syntezator mowy. Mogą one działać oddzielnie, pozwalając na edycję poszczególnych elementów tworzących język. Schemat ogólnych oddziaływań między modułami podczas rozpoznawania języka przedstawia Rys. 1.2.2

System składa się z trzech modułów rozpoznających budowę języka, oraz syntezatora mowy, oznacza to potrzebę stworzenia czterech programów, które współdzielą ze sobą bazy danych zapisane na dysku twardym. Rozmieszczenie poszczególnych katalogów zawierających dane dla aplikacji (przedstawione na Rys.1.2.3), zostało zaplanowane tak, by stało się czytelne zarówno dla maszyny jak i użytkownika. W folderze głównym znajdują się programy wykonywalne, są to; ”Syntezator”, ”Sampler”, ”Kreator” oraz ”Strukturyzator”. Został w nim także utworzony folder z danymi potrzebnymi do działania wymienionych programów, zawierający pliki graficzne oraz pliki wymiany. Ważnym katalogiem jest katalog ”sample”. Zostały w nim umieszczone kolejno: ”baza.txt” (spis wszystkich baz językowych), oraz (jeśli istnieją bazy językowe) pliki z rozszerzeniem *.synt i katalogi o tych samych nazwach. Wewnętrzna budowa pliku *.synt oparta jest na standardach plików informacyjnych

*.ini systemu Windows™. Pliki w tym katalogu należy wskazać syntezatorowi w trakcie wyboru bazy językowej, podczas którego syntezator pozwala jedynie na wybór plików o rozszerzeniu *.synt. Zawiera on informacje konfigurujące syntezator oraz wskazuje miejsce, w jakim przechowywane są sample (katalog o tej samej nazwie, co plik *.synt) i baza struktur. Ponadto zapisano w nim między innymi czas trwania przerwy pomiędzy poszczególnymi wyrazami, a także przerwy pomiędzy kolejnymi próbkami dźwięków (podawane w mili sekundach). Poniżej znajduję się budowa przykładowego pliku o nazwie ”fonemy_pl.synt”.

```
[Sample]
sciezka=fonemy_pl//
[Odczyt]
predkosc_odczytu=186
czas_spacji=100
```

Katalog o tej samej nazwie, co plik *.synt przedstawiony na Rys. 1.2.3 zawiera próbki językowe (sample) zapisane w formacie plików dźwiękowych o rozszerzeniu *.wav. Nazwa pliku zawierającego próbkę dźwiękową jest komplementarna z jego odpowiednikiem znakowym. W tym samym katalogu umieszczono ”index.txt”. Zawiera on spis wszystkich nagranych sampli, które znajdują się w bazie, oraz plik ”do_nagrania.txt” posiadający informacje dla programu ”Sampler”, które z wzorców znakowych oczekują jeszcze na przypisanie im odpowiednika dźwiękowego. Plik ten będzie wypełniony danymi zaraz po zakończeniu pracy w programie ”Kreator”. Jego zawartość będzie się zmniejszać wprost proporcjonalnie do tego jak zwiększać się będzie rozmiar pliku ”index.txt” w trakcie działania programu ”Sampler”. W chwili, gdy wszystkie wzorce znakowe będą posiadały swoje dźwiękowe odpowiedniki plik ”do_nagrania.txt” będzie pusty, a jego zawartość zostanie przepisana do ”index.txt”. W przypadku usunięcia sampli (funkcja dostępna w programie ”Sampler”), przepływ danych będzie zwrócony w odwrotnym kierunku. Wzorce znakowe oczekujące na przypisanie im odpowiedników dźwiękowych można trwale usuwać lub dodawać (w programie ”Sampler”). Kolejnym plikiem znajdującym się w omawianym katalogu są ”struktury.dat”, zawierający dane zgodne ze standardem zapisu danych w systemie Windows™. Informacje (przedstawiające struktury powiązań językowych) zapisane w tym formacie nie mogą podlegać edycji w innych programach, niż te z pakietu ”SYNT”.



Rys. 1.2.3 Rozmieszczenie na dysku danych i programów pakietu ”SYNT”.

Struktury są skróconym zapisem znakowym, jaki może zostać zapisany znakowo w sposób rozwinięty. Obydwa zapisy posiadają taki sam odpowiednik werbalny. Struktury można podzielić pod względem ich złożoności.

Wybrane struktury proste:

- Podstawowe znaki, z jakich zbudowany jest alfabet np. ”A”,”b”,
- skróty językowe np. ”m.n.p.m”, ”itp.”,
- cyfry od zera do granicy ich powtarzalności (do przyrostka ”naście”) i niektóre cyfry rozpoczynające kolejne jednostki rzędu decy ”dwadzieścia”, ”pięćdziesiąt”, ”sto”.

Wybrane struktury złożone:

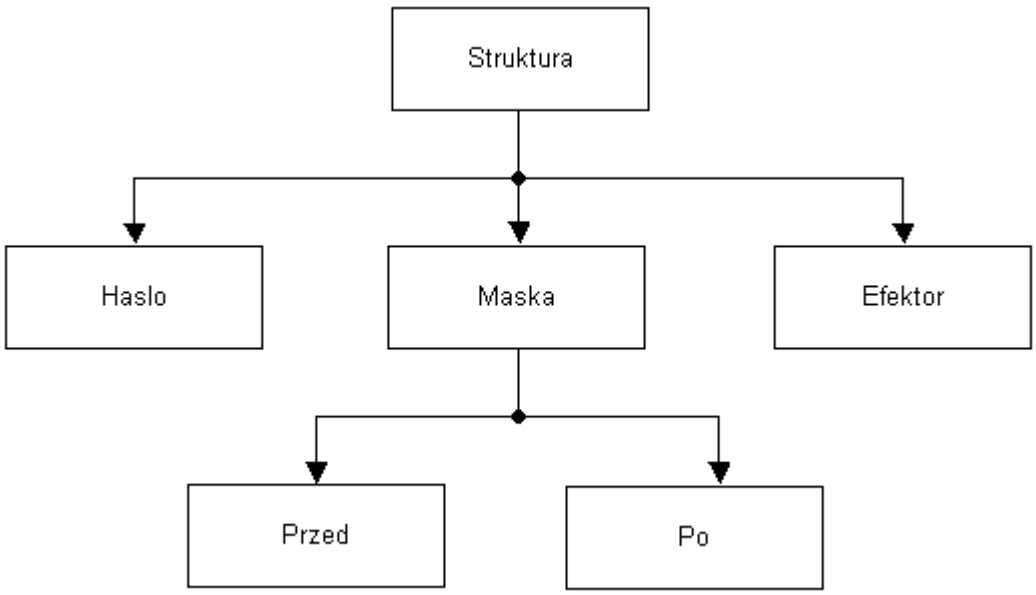
- Cyfry od granicy ich powtarzalności (od przyrostka ”naście”),
- uproszczenia językowe np. ”isn’t” oznaczające ”is not”.

Często struktury proste stanowią element budowy zarówno struktur prostych jak i złożonych. Struktury proste posiadają przypisane odpowiedniki dźwiękowe, natomiast złożone zostają zdefiniowane za pomocą struktur prostych.

W przypadku numeru 130, należy stworzyć strukturę złożoną. W skład przedstawionej cyfry wchodzą struktury proste: ”1”, ”3” i ”0”, które posiadają odpowiednik werbalny brzmiący ”jeden”, ”trzy”, ”zero”. Jednak z uwagi na ich rozmieszczenie względem siebie, ich odpowiednikami struktur prostych będą ”100”, ”30”, ”NULL”, natomiast ich werbalne odpowiedniki to ”sto”, ”trzydzieści”, ”␣”

1	3	0
Przedrostek (prefiks)	Wrostek (infiks)	Przyrostek (sufiks)

Strukturę złożoną można opisać przestrzennie. Zgodnie z przedstawionym przykładem liczba ”3” jest infiksem, ”1” prefiksem natomiast ”0” sufiksem. Z uwagi na to, że strukturami złożonymi nie są tylko liczby, algorytm nie opiera się na sprawdzaniu rzędu jedności, dziesiątek, setek i wyższych rzędów. Po wprowadzeniu tekst zostaje on umieszczony w tablicy, w której znaki następują kolejno po sobie. Analizę można rozpocząć po uprzednim zdefiniowaniu budowy struktury. Rys.1.2.4. W skład struktury wchodzi ”hasło”, ”maska” i ”efektor”, natomiast maska zawiera definicję czy występuje ona ”przed” czy ”po” ”haśle” Rys. 1.2.4.



Rys. 1.2.4 Budowa struktury

Przykład z Rys. 1.2.5 pokazuje strukturę złożoną ”21”, która została umieszczona w tablicy statycznej N zakończonej w N[8] a rozpoczynającej się w N[1], struktura zajmuje pozycje od N[5] do N[6]. Algorytm analizujący struktury przegląda tablicę od jej początku aż do końca.

E	L	A		2	1		

Rys. 1.2.5 Tablica statyczna N wypełniona przykładowym tekstem

W przypadku napotkania struktury (znaku) sprawdzane jest czy spełnia ona warunki struktury złożonej. Sprawdzona zostaje zgodność elementu tablicy z hasłem struktury, w przypadku jej stwierdzenia analizowana jest maska. Maska definiuje ile znaków (struktur prostych) może występować po haśle. Gdy zależność ta nie zwraca negacji, sygnalizuje to wykrycie struktury złożonej. Jej efektor zostaje wpisany od miejsca jej wykrycia a rozmiar tablicy statycznej jest zmieniany na nowy, powiększony o długość ”efektora”. Przykład z Rys. 1.2.5 przedstawia definicje struktur zamieniających ”21” na ”dwadzieścia” ”jeden” jest ona zrealizowana w sposób przedstawiony poniżej:

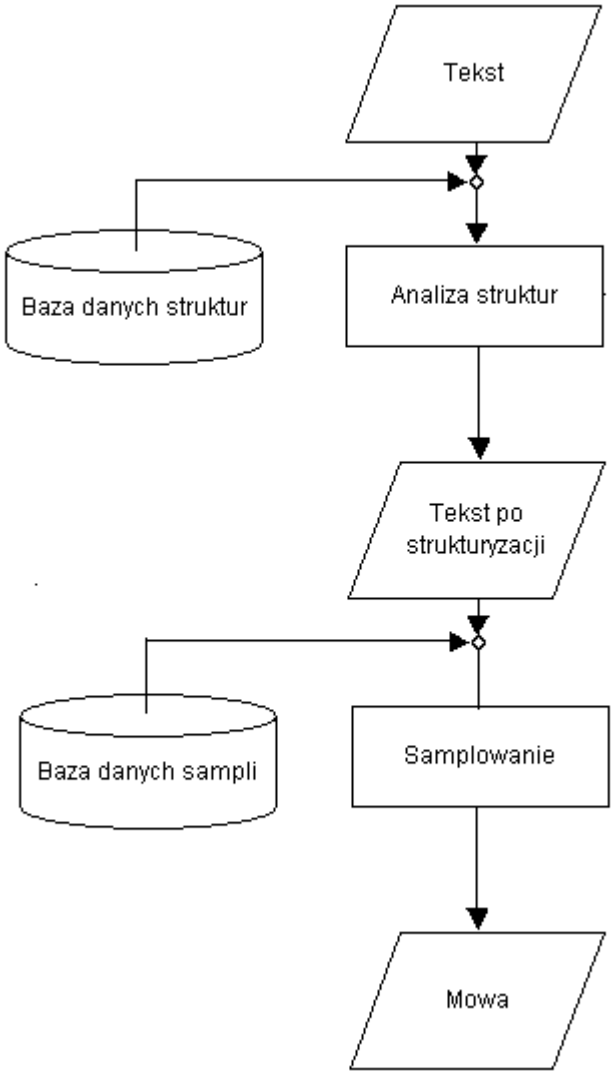
Struktura złożona ”dwadzieścia”:
Przed=false (maska nie występuje przed hasłem)
Po=true (maska występuje po haśle)
Maska: # (co oznacza jeden dowolny znak)
Hasło: 2 (po napotkaniu tego znaku analizowana jest struktura złożona)
Efektor: ”dwadzieścia␣” (zakończony spacją)

Struktura prosta ”jeden”:
Przed=false (ponieważ brak maski)
Po=false (ponieważ brak maski)
Maska: NULL (brak)
Hasło: 1 (po napotkaniu tego znaku zamieniany jest on na efektor)
Efektor: ”jeden␣” (zakończony spacją)

Struktury operują morfemami, analizują prefiksy, sufiksy i interfiksy na tablicach znaków, gdzie pojedynczy element tablicy zawiera dokładnie jedną strukturę prostą, jaka w relacji z otaczającymi ją innymi strukturami prostymi może tworzyć strukturę złożoną.

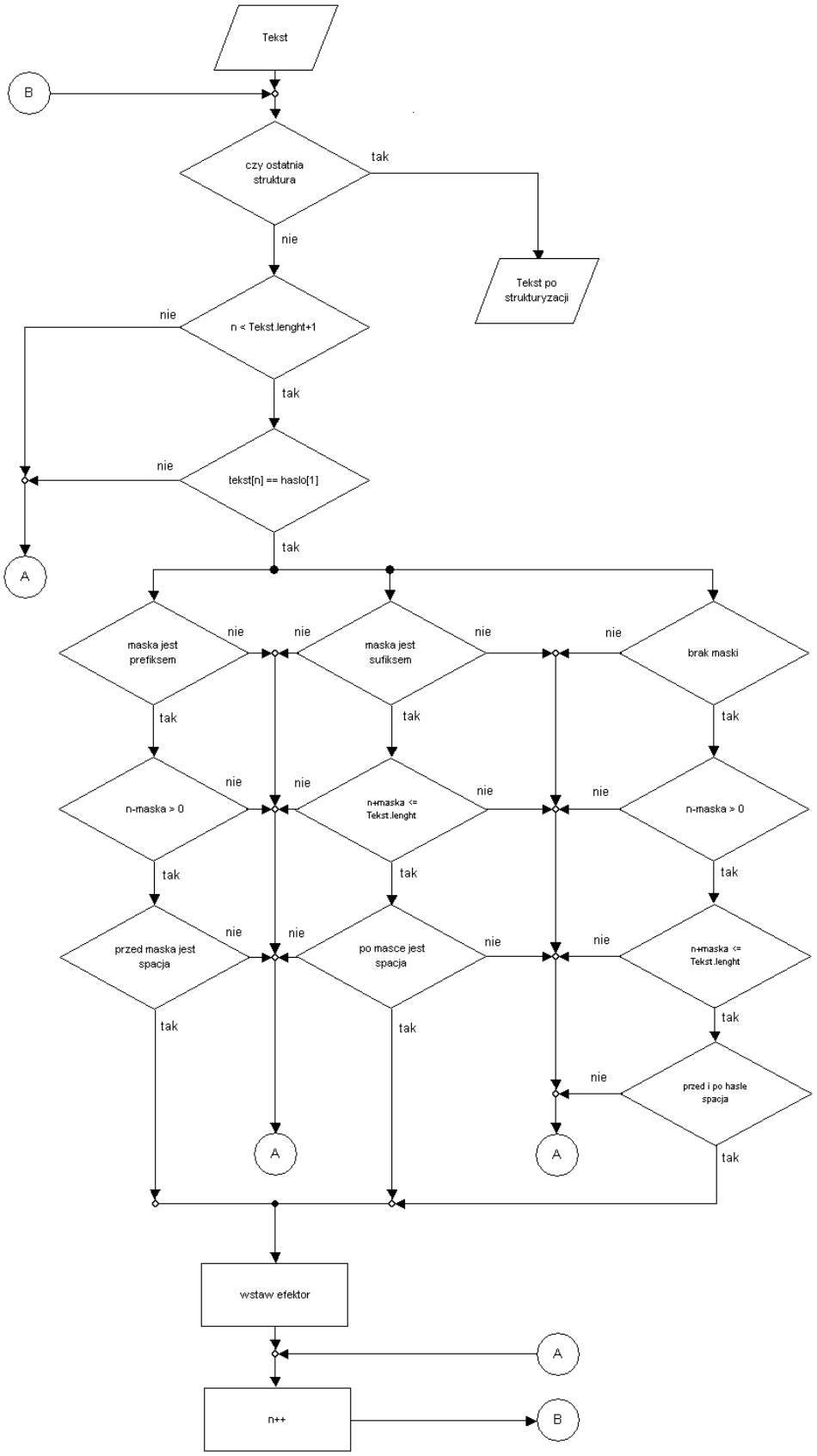
Algorytm powstawania mowy, oparty na metodzie konkatencyjnej został przedstawiony na

Rys.1.2.1. Do przeprowadzenia procesu syntezy TTS potrzebne są następujące substraty: ”Baza danych sampli”, ”Baza danych struktur”, ”Baza danych alfabetu” oraz tekst w zdefiniowanym języku. Z substratów zostaje otrzymany produkt - mowa Rys. 1.2.6. Proces syntezy odbywa się etapowo. Niemożliwe jest powiązanie wszystkich elementów podczas trwania jednego kroku, ponieważ zakłóci to działanie całego procesu. Analiza struktur Rys. 1.2.7 operuje na znakach, a jako produkt otrzymuje się znaki Rys.1.2.6. W kolejnym etapie na przetransformowanym tekście poczynione zostaje przypisywanie próbek dźwiękowych do ich znakowych odpowiedników, efektem jest werbalizacja Rys.1.2.8.



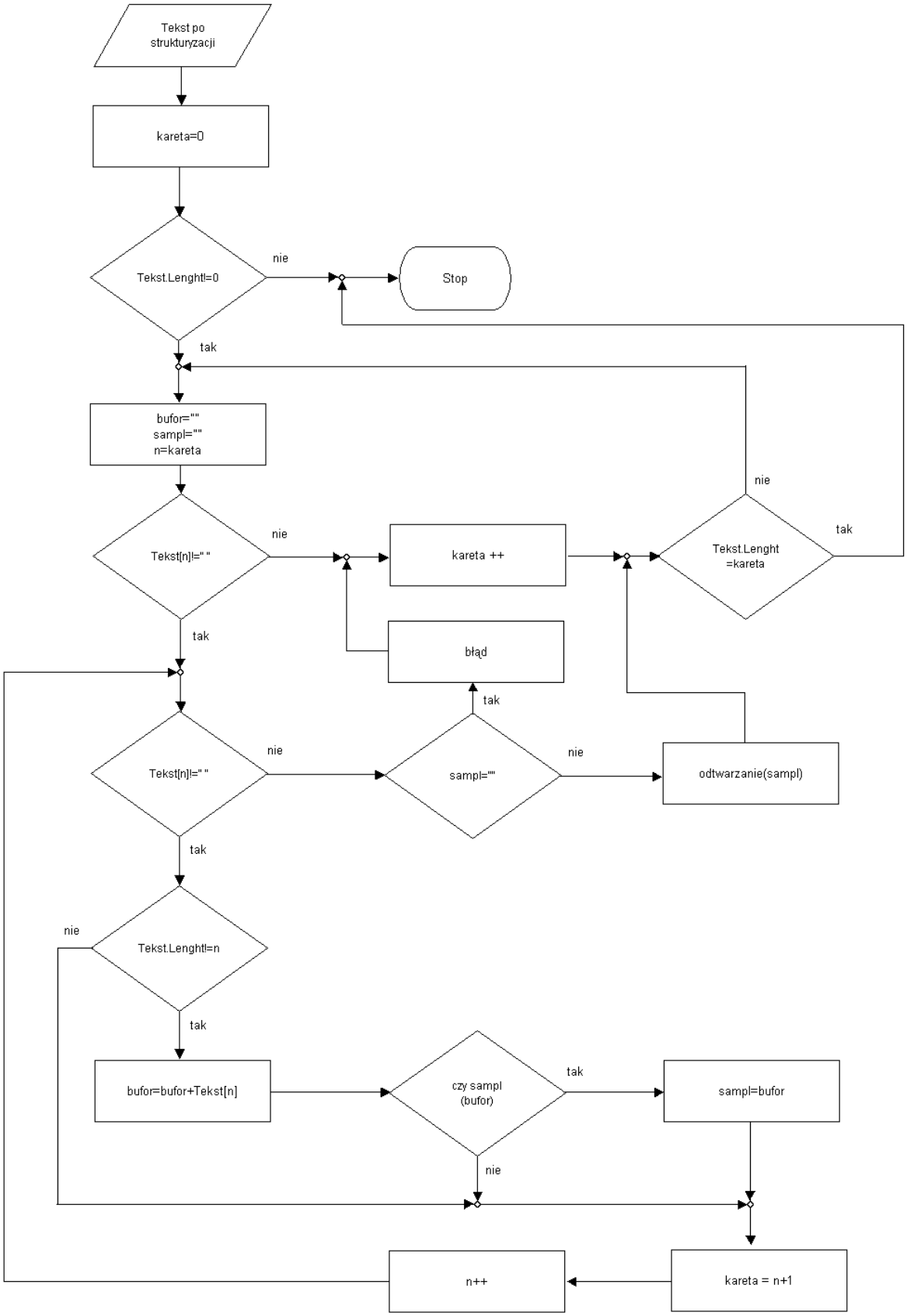
Rys. 1.2.6 Ogólny schemat opracowanego algorytmu TTS

Analizę struktur przedstawia Rys. 1.2.7. Operuje ona na bazie, która zawiera pojedyncze struktury. Budowę pojedynczej struktury przedstawia Rys. 1.2.4. Algorytm Rys. 1.2.7 sprawdza ilość posiadanych w bazie rekordów, pod kątem każdego z nich analizuje wprowadzony tekst. Po pobraniu z bazy jednej struktury, analizowane są kolejne znaki tekstu do chwili napotkania ostatniego znaku. Kolejne znaki poddawane są komparacji z ”hasłem” struktury i w przypadku wystąpienia ciągu we wprowadzonym tekście, jaki odpowiada na ”hasło” wykonywana jest dokładna analiza. Zostają sprawdzone maski struktury za pomocą wirtualnej injekcji analitycznej (przestrzenne potwierdzenie istnienia maski i hasła w tekście). Po otrzymaniu informacji potwierdzającej obecność zdefiniowanej ”maski” w miejscu analizowanego tekstu wstawiany jest ”efektor”. Po dokonaniu rozpoznania struktury lub wykazania jej braku, algorytm przechodzi do kolejnego znaku tekstu, jaki nie został jeszcze przeanalizowany. W chwili, gdy tablica znaków wprowadzonego tekstu zostanie cała przeanalizowana, algorytm pobiera kolejną strukturę z bazy i proces powtarza się do momentu sprawdzenia tekstu z udziałem wszystkich struktur z bazy. Analizowany tekst jest tablicą statyczną w trakcie trwania analizy, nowa tablica powstaje po rozpoznaniu struktury i wpisaniu zawartości poprzedniej tablicy, oraz ”efektora” struktury. Zmienność tekstu wymaga ciągłego sprawdzania wielkości tablicy. Zasada ta uniemożliwia analizowanie wszystkich struktur w jednym kroku. Po wykonaniu algorytmu, tekst zawiera struktury proste, jakie posiadają odpowiedniki dźwiękowe. Kolejnym etapem jest transformacja TTS.



Rys. 1.2.7 Schemat procesu analizy struktur

Proces przypisywania sampli, nazywany ”Text To Speech” przedstawia Rys. 1.2.8. Czas procesora jest zajmowany przez algorytm TTS krócej niż przez algorytm analizy struktur. Wykonywanych zostaję mniej analiz i pętli, mniejsze jest też użycie baz danych. Algorytm otrzymuje jako substrat tekst po strukturyzacji (lub tekst zwykły w przypadku, gdy analiza struktur nie została wykonana). Po ustawieniu karety (pozycja odczytu) na początek tekstu, zostaje on sprawdzony, czy zawiera znaki. W przypadku, gdy ich nie posiada algorytm zostaje zatrzymany. Wartość zmiennej ”*kareta*” przepisywana zostaje do zmiennej ”*n*”. Zmienna ”*bufor*” jest to ciąg pomocniczy, w którym przechowywany jest analizowany tekst podczas jednej operacji przeszukiwania. Natomiast ”*sampl*” to zmienna ze znalezionym najdłuższym buforem, posiadającym odpowiednik w bazie sampli. Zmienne ”*bufor*” oraz ”*sampl*” są zerowane. Pobierany zostaje ”*n*-ty” element tablicy tekstu. Jeśli nie jest spacją oznacza, że rozpoczyna się wyraz i poszukuje się najdłuższego możliwego wzorca dźwiękowego. Operacja polega na wykonywaniu pętli aż do napotkania kolejnej spacji – końca wyrazu. Pętla wykonuje się od pozycji karety równej ”*n*” do pozycji ”*n+x*” gdzie ”*x*” oznacza pozycję kolejnej spacji. Sprawdzane zostają wartości granic tablicy tekstu. Bufor uzupełnia się o kolejny znak analizowanej tablicy tekstu. Gdy zawartość znakowa bufora znajdzie swoje odzwierciedlenie w bazie sampli, do zmiennej ”*sampl*” przypisywana jest wartość bufora. ”*Kareta*” znajduje się w aktualnej pozycji ”*n*”. Poszukiwanie ”sampla” odbywa się od początku. Pobierany jest kolejny znak i zostaje sprawdzone czy dłuższa kombinacja znajdzie odzwierciedlenie w bazie sampli. Zadaniem algorytmu jest znalezienie możliwie najdłuższego ciągu znaków, jaki posiada odpowiednik w bazie sampli (komplementarność). Algorytm samplowania w początkowej wersji posługiwał się znaną długością sampli (na przykład samplowanie po jednym znaku, lub po dwa). Na obecnym etapie rozwoju poszukuje najlepszych rozwiązań.

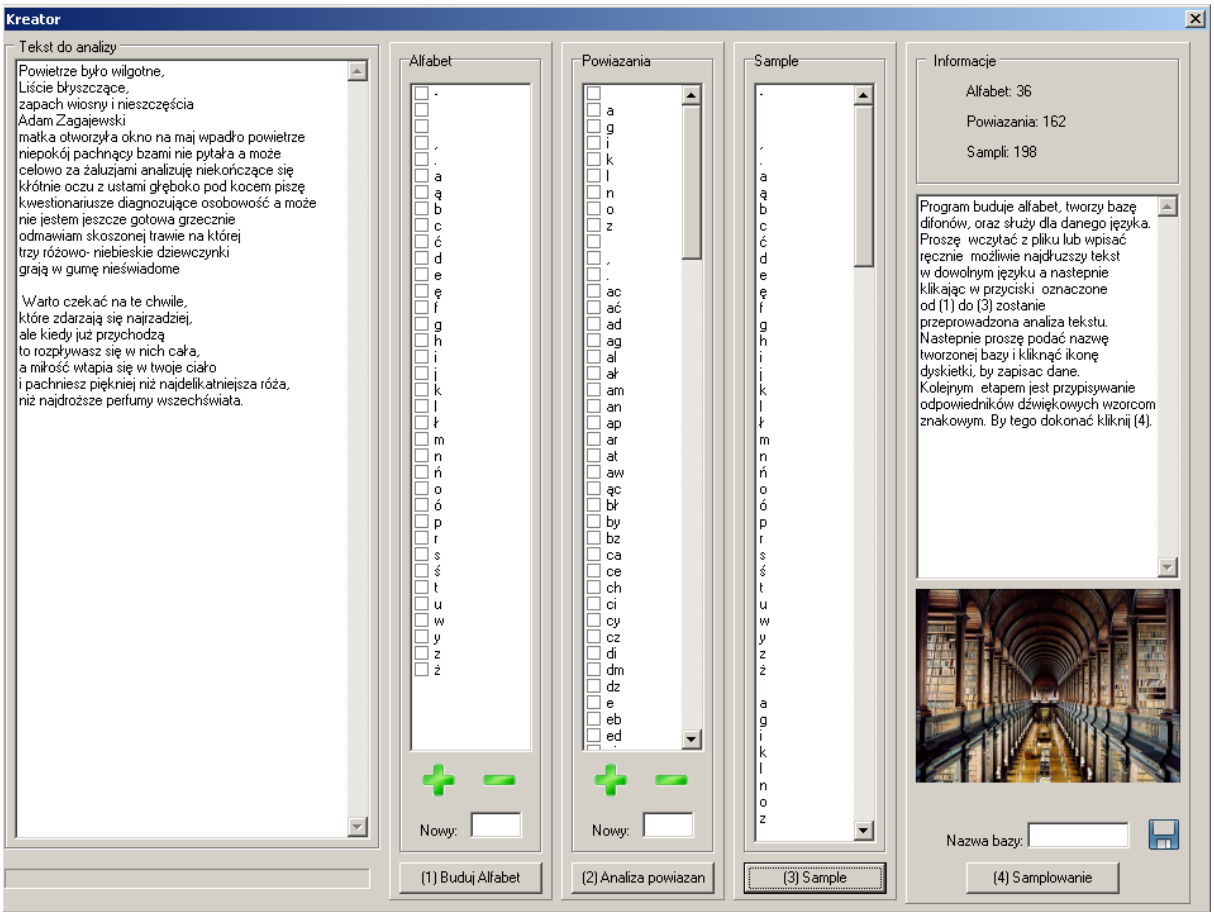


Rys. 1.2.8 Schemat procesu Samplowania - powstawanie mowy

II. Opis funkcjonowania programu

Algorytmy omawiające problematykę rozdziału [1.2] zostały zaprogramowane w ”C++ Builder”, powstały zbiór aplikacji nazwano ”SYNT”. Został on wyposażony w narzędzia, opisane w rozdziale [1.2]. Na Rys.1.2.2 przedstawiono relacje pomiędzy modułami analizującymi budowę języka, w tej kolejności zostaną przedstawione aplikacje, ich obsługa, oraz fragmenty kodu.

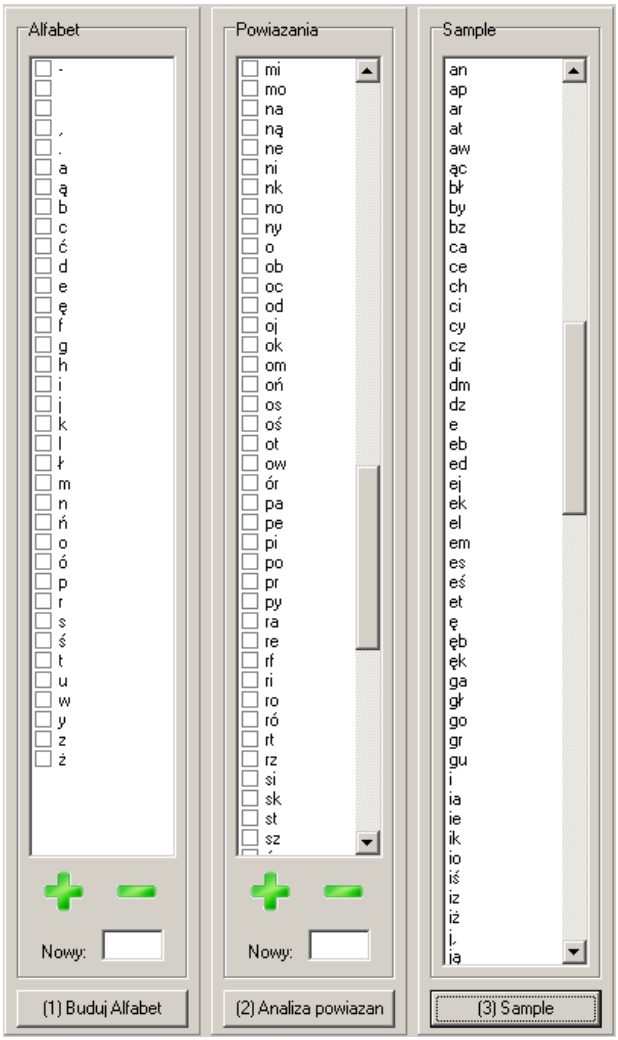
II.1 Kreator



Rys. 2.1.1 ”Kreator” do analizy języka, program pakietu ”SYNT”

Pierwszym programem pakietu ”SYNT” jest ”Kreator”. Praca w z nim rozpoczyna się od wpisania lub wczytania możliwie jak najdłuższego tekstu w analizowanym języku, jest on widoczny w polu ”Tekst do analizy”. Zadaniem aplikacji jest przeprowadzenie tekstu przez szereg filtrów językowych, jakie tworzą alfabet, a następnie analizują proste powiązania językowe, jak np. ”rz” ”cz” (tworzenie difonów). Ostatni filtr buduje listę znaków, którym zostanie przypisany wzorzec dźwiękowy. W programie istnieje możliwość zdefiniowania

struktur językowych (jednak nie jest konieczna na tym etapie analizy i można je uzupełnić w programie ”Strukturyzator”).

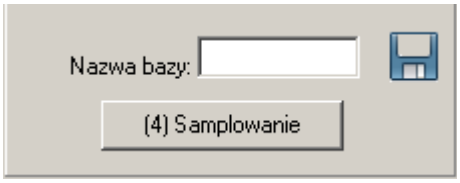


Rys. 2.1.2 *Obiekty klasy TcheckListBox*

Pierwszy filtr tworzący alfabet uruchamiany zostaje po naciśnięciu przycisku ”(1) Buduj Alfabet”. W komponencie klasy TcheckListBox zostaje wyświetlony rezultat. Istotną sprawą jest by usunąć wszystkie nieodpowiednie znaki nie należące do alfabetu. Można tego dokonać zaznaczając je na liście i klikając w ”zielony minus”. Lub dodać brakujące, po wpisaniu znaku w polu ”nowy” i klikając w ”zielony plus” Rys. 2.1.2. ”(2) Analiza powiązań” uruchamia filtr powiązań. Powstałe elementy zostają również wyświetlone w TcheckListBox i podobnie jak w przypadku alfabetu można dodawać nowe oraz usuwać już istniejące Rys. 2.1.2. ”(3) Sample”, jest to ostatni filtr tworzący listę sampli. Efekt wyświetla się w komponencie TlistBox Rys. 2.1.2. Użytkownik nie ma możliwości edycji powstałej listy.

Kod w języku C++ pobierający alfabet z tekstu przedstawia ”załącznik 1”, analiza powiązań została rozwiązana w sposób przedstawiony w ”załączniku 2”. Funkcje jakie zostały użyte we wspomnianych algorytmach zawiera ”załącznik 3”.

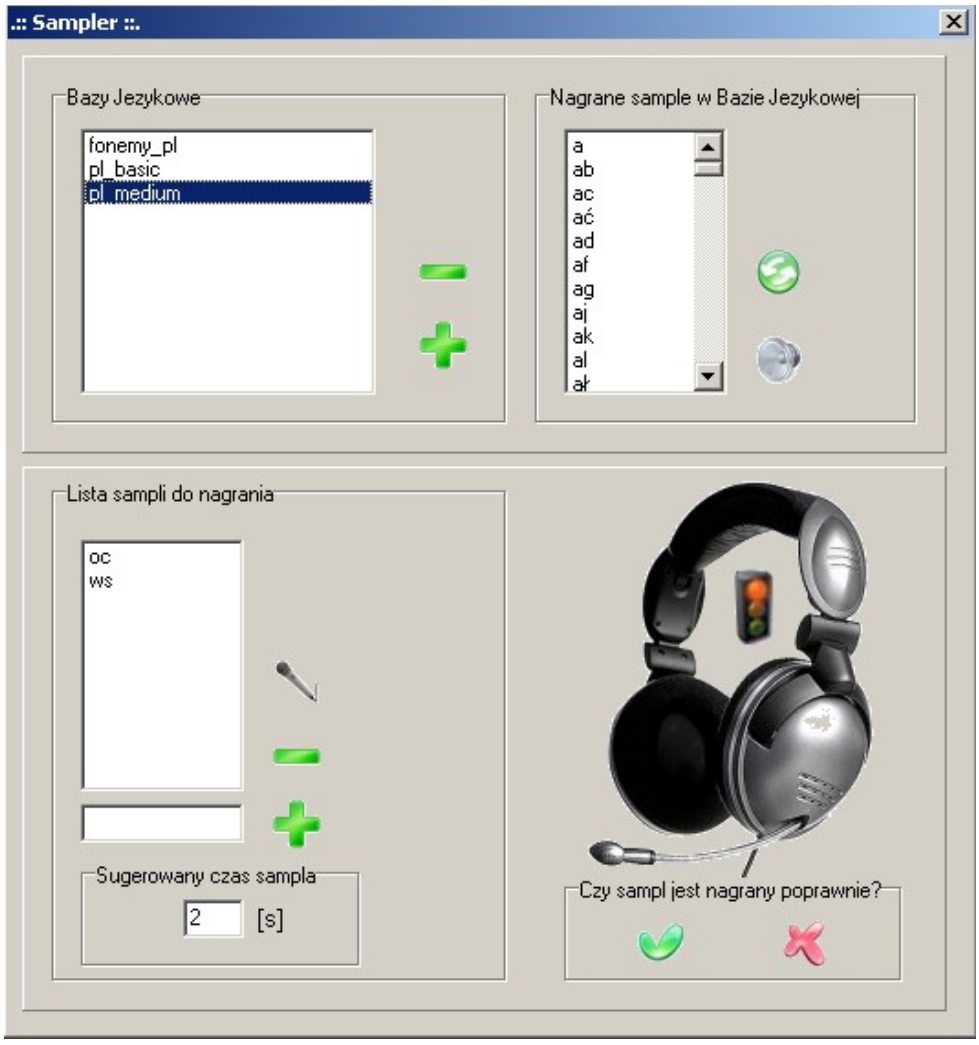
Po zakończeniu rozpoznania budowy języka, gdy wykonano analizę tekstu przy użyciu wszystkich filtrów językowych od (1) do (3) pokazanych na Rys. 2.1.2, należy zapisać projekt jako nową bazę językową. Nazwę bazy podaje się w polu Tedit, a następnie w celu jej zapisania należy nacisnąć ikonę dyskietki Rys. 2.1.3. Kolejny etap to przypisywanie wzorców dźwiękowych do stworzonych wzorców znakowych. Przejście do aplikacji samplującej nastąpi po kliknięciu w przycisk ”(4) Samplowanie”.



Rys. 2.1.3 Utworzenie bazy językowej

II.2 Sampler

Zadaniem programu ”Sampler” jest nagranie próbek (wzorców) językowych i przypisanie ich odpowiednikom znakowym, które zostały stworzone w programie ”Kreator”. Próbki są zapisane jako baza językowa, która umożliwia odczyt języka przez syntezator. Baza sampli może być nagrywana, edytowana oraz usuwana w programie ”Sampler”. Istnieje również możliwość dodania nowych sampli, gdy ich potrzeba zostanie wykazana w trakcie pracy aplikacji ”Syntezator”. Obsługa procesu przypisywania próbek dźwiękowych do ich odpowiedników znakowych przeprowadzana jest przez użytkownika. W panelu ”bazy językowe” przedstawionym na Rys. 2.2 widoczne są aktualnie dostępne języki, które mogą być analizowane przez ”Syntezator”. Zielony ikona ”plus” umożliwia dodanie nowych baz językowych(uruchamia program ”Kreator”), natomiast ikona ”minus” usuwa wybraną bazę. Dział ”lista sampli do nagrania” zawiera sample, które wymagają przypisania odpowiednika dźwiękowego. Sample mogą zostać usunięte z listy po ich podświetleniu i kliknięciu w ikonę ”minus”.



Rys. 2.2 Program ”Sampler” z pakietu ”SYNT” do nagrywania i edycji sampli

Natomiast dodanie nowego wzorca znakowego następuje po jego wprowadzeniu z klawiatury (pole obok ikony ”plus”) i kliknięciu w ikonę ”plus”. Przypisanie odpowiednika dźwiękowego wykonywane jest po podświetleniu wybranego wzorca znakowego. Czas trwania nagrania zostaje analizowany przez program, istnieje możliwość jego zmiany w panelu ”sugerowany czas sampla”. Jednostką miary jest sekunda. Gdy zaistnieje konieczność jego korekty, należy to zrobić przed rozpoczęciem nagrywania. Nagrywanie wzorca dźwiękowego rozpoczyna się po naciśnięciu ikony ”mikrofonu”. W prawej części okna przedstawionym na Rys. 2.2 znajduje się obraz słuchawek a w nim ”światła sygnalizacyjne”. Zapalają się one kolejno w kolorach; czerwonym, żółtym (przygotowanie do nagrywania), zielonym (nagrywanie). Zaraz po rozpoczęciu nagrywania należy wymówić do mikrofonu dźwięk reprezentujący głosowy odpowiednik wybranego wzorca znakowego. Po zakończeniu procesu nagrywania jest on odtwarzany. Jeśli został zarejestrowany poprawnie, potwierdza się to kliknięciem w zieloną ikonę ”Tak”, w przypadku nieprawidłowego samplowania program informuje się o tym klikając czerwoną ikonę ”Nie”. Poprawnie nagrany dźwięk na podstawie

wzorca znakowego staje się samplem, który jest używany przez program ”Syntezator”. Sampl przenoszony zostaje do panelu ”nagrane sample w bazie językowej” Rys. 2.2. Podświetlenie sampla w panelu ”nagrane sample w bazie językowej” i klikniecie ikony głośnika, powoduje jego odtworzenie, natomiast zielona ikona służy do usuwania odpowiednika dźwiękowego i przeniesienia wzorca znakowego na ”listę sampli do nagrania”.

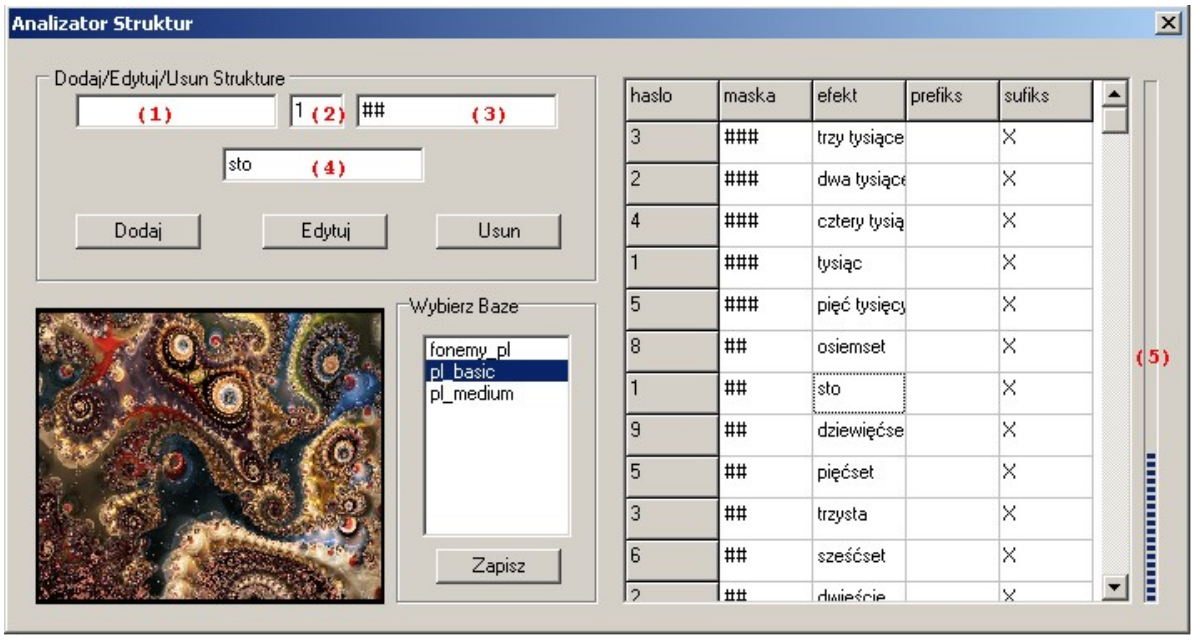
Obsługa mikrofonu została zrealizowana przy pomocy specjalnie napisanej do tego celu biblioteki. Sample zapisywane są w formacie WAVE. Struktura typu WAVEFORMATEX na potrzeby funkcji ”waveInOpen” została wypełniona jak na podanym poniżej przykładzie.

```
waveformatex.wFormatTag = WAVE_FORMAT_PCM; //Pulse Code Modulation (PCM),
                                           //not compression
waveformatex.nChannels = 1; //channels
waveformatex.nSamplesPerSec = 11025; //samples per second
waveformatex.nAvgBytesPerSec = 11025; //bites per second
waveformatex.nBlockAlign = 1;
waveformatex.wBitsPerSample = 8; //bites per sample
waveformatex.cbSize = 0; //extra information
```

W celu zmiany jakości dźwięku sampli trzeba zmienić powyższe parametry struktury.

II.3 Analizator struktur

Analizator struktur jest aplikacją do edycji struktur językowych, zostały one omówione w rozdziale [1.2], gdzie przedstawiono przykłady ich użycia, oraz budowę. Budowę pojedynczej struktury, z jakich składa się cała tablica struktur przedstawia również ”załącznik 4”. By dodać nową strukturę należy podać ”maskę” (ilość znaków przed lub po ”haśle” oznaczonych symbolem ”#” dla jednego znaku). W przypadku, gdy ”maska” występuje przed ”hasłem” wpisana zostaje w pole (1) oznaczone na Rys. 2.3, analogicznie dla ”maski” po ”haśle” zostaje ona umieszczona w polu (3) Rys. 2.3. ”Efektor” należy podać w polu (4) Rys. 2.3. Struktury proste jak (na przykład ”m.n.p.m”), jakie nie wchodzą w skład struktur złożonych należy nagrać w programie ”Sampler”.

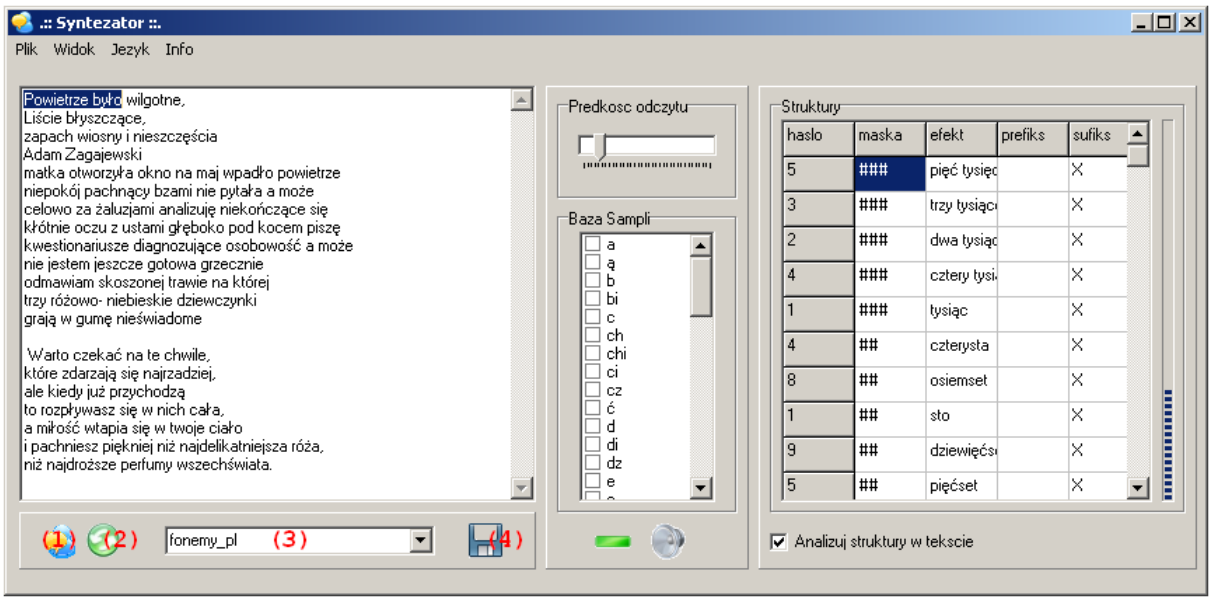


Rys. 2.3 Analizator struktur, program z pakietu ” SYNT”

Baza struktur jest bazą o rozmiarze statycznym (ograniczonym objętościowo). Przeznaczono w niej miejsce na zapisanie maksymalnie stu rekordów. Pasek informujący o poziomie zapełnienia bazy został umieszczony obok wyświetlonych rekordów, oznaczony jako (5) na Rys. 2.3. Po podświetleniu wybranej bazy z danymi (panel ”Wybierz bazę”) istnieje możliwość dodawania i usuwania zasad struktury (panel ”Dodaj/Edytuj/Usuń struktury” Rys. 2.3) klikając przycisk ”Dodaj”, lub ”Usuń” (po ich uprzednim wybraniu z wyświetlonych rekordów). Wszystkie zmiany dokonane podczas edycji wybranej bazy językowej, zostają zapisane po kliknięciu w przycisk ”Zapisz” (panel ”Wybierz bazę” Rys. 2.3).

II.4 Syntezator

Syntezator jest głównym programem pakietu ”SYNT” z punktu widzenia użytkowego. Wykorzystuje on wszystkie zbudowane we wcześniej omówionych programach elementy języka. Efektem działania TTS jest mowa werbalna odtwarzana przy użyciu karty dźwiękowej komputera lub modemu. Przed rozpoczęciem pracy w programie należy wczytać bazę sampli wybrając ją z rozwijalnej listy oznaczonej jako (4) na Rys. 2.4. Program wyposażony jest w opcję odczytu plików tekstowych zapisanych na dysku twardym, lub też na innym nośniku.

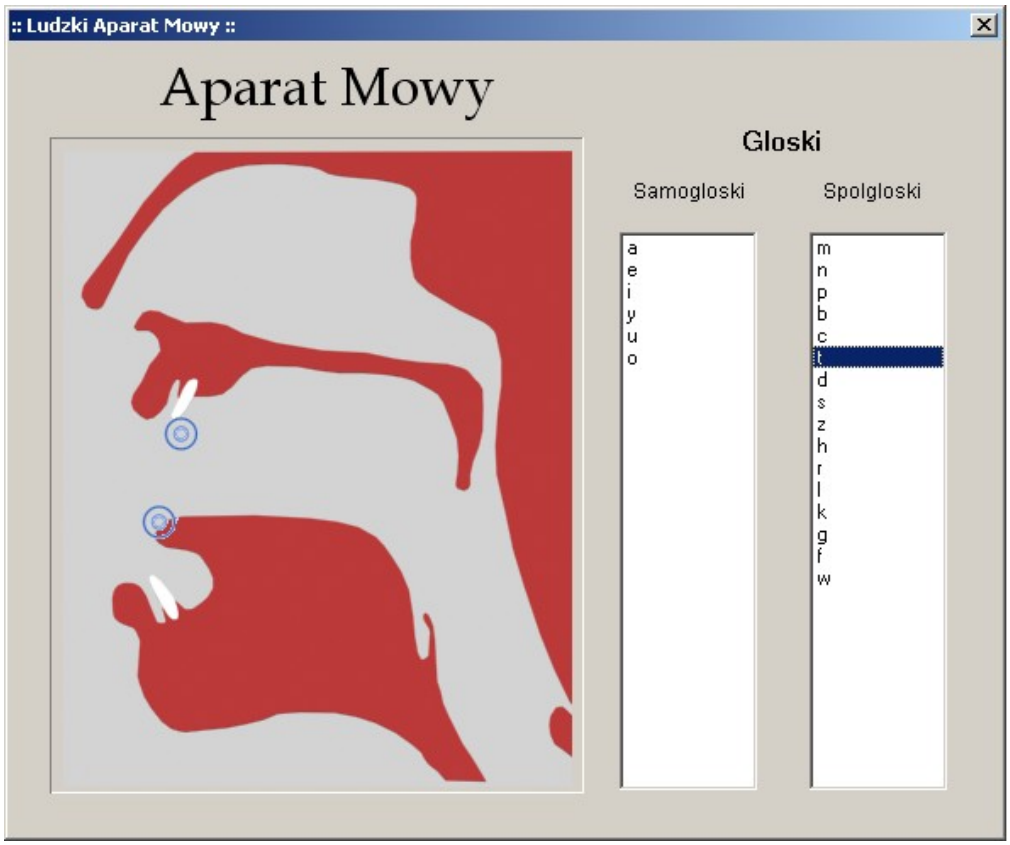


Rys. 2.4 Syntezator mowy, z pakietu ”SYNT”

Po wybraniu w menu ”Plik” i ”Otwórz z pliku” należy wskazać tekst w formacie *.txt i go wczytać. W trakcie działania programu istnieje możliwość wysłania komunikatów do odczytu z innych aplikacji. Tryb ten wiąże się z ciągłą pracą programu, by nie przeszkadzał on w wykonywaniu innych operacji, aplikacja po zminimalizowaniu znajduje się obok zegarka (pośród takich ikon jak połączenie sieciowe, regulacja głośności itp.), tak zwany pasek ”tray”. Odczyt tekstu, jaki nie został wysłany z innej aplikacji, następuje po kliknięciu ikony oznaczonej jako (1) na Rys. 2.4. Sąsiadująca ikona (2) przedstawiona na Rys. 2.4 czyści pole tekstu do odczytu. Syntezator został wyposażony w dwa tryby obsługi; ”prosty” i ”zaawansowany”. Przełączenia można dokonać w menu ”Widok”. Różnią się one nie tylko wielkością okna syntezatora, ale i możliwościami podglądu struktur językowych, bazy sampli, i prędkości odczytu. Tryb ”zaawansowany” daje możliwość odsłuchania pojedynczych sampli a nawet ich czasowego usunięcia. Ustawienia syntezatora zapisywane są automatycznie po jego wyłączeniu. Przy ponownym uruchomieniu otworzy się on w takim trybie, w jakim został zamknięty. Ustawienia dotyczące bazy danych językowych jak na przykład prędkość odczytu sampli, można zapisać klikając ikonę ”dyskietki” (4) pokazaną na Rys. 2.4. Ustawienia te zostaną wczytane przy ponownym otworzeniu bazy językowej. Algorytm pętli czytającej Rys. 1.2.8 przedstawia ”załącznik 5”. Natomiast pętle sprawdzającą czy w schowku nie ma polecenia z tekstem do odczytu zamieszono w ”załączniku 6”.

II.5 Ludzki Aparat Mowy

Program powstał w trakcie pracy nad pakietem ”SYNT”. Pokazuje on, w jakich rejonach ludzkiego aparatu mowy powstają dźwięki głosek - najmniejszych elementów dźwiękowych form wypowiedzi. Głoski charakteryzują się stałym zespołem fonetycznych cech artykulacyjnych i akustycznych. Każda głoska jest fizyczną realizacją jakiegoś fonemu, który może mieć więcej niż jedną realizację. Wyróżnia się trzy fazy głoski: następ (przygotowanie), szczyt (przyjęcie przez narządy mowy pozycji właściwej danej głosce) i zestęp (powrót do pozycji neutralnej). Ze względu na różnice akustyczne, fonetyczne i funkcjonalne głoski dzielą się na spółgłoski i samogłoski. Samogłoski są czystymi tonami, podczas gdy spółgłoski zawierają element szmerowy. Ponadto samogłoski mają zdolność tworzenia sylab. Podczas artykulacji samogłosek powietrze przepływa przez otwarty kanał, nie napotykając na przeszkody jak w przypadku spółgłosek.[5] Po zgromadzeniu informacji o rodzajach samogłosek, spółgłosek i ich podziału została utworzona aplikacja przedstawiona na Rys. 2.5.



Rys. 2.5 Program przedstawiający działanie ludzkiego aparatu mowy

[5] pis.por. http://pl.wikipedia.org/wiki/Język_polski#Fonetyka

Procesy powstawania dźwięku, zaprezentowano za pomocą prostej animacji. Po kliknięciu w wybraną głoskę z kolumny samogłosek lub spółgłosek (przedstawionych na Rys.2.5) następuje odtworzenie jej odpowiednika dźwiękowego i zaznaczenie na anatomicznym obrazie ludzkiego traktu głosowego.

”Trakt głosowy, zwany też kanałem głosowym, dokonuje artykulacji mowy. Artykulacja to wszystkie akcje wymienionych wyżej organów, prowadzące do powstania dźwięku, a w rezultacie mowy. Źródłem energii dla sygnału mowy są płuca. Dostarczają one powietrza do procesu artykulacji. W czasie wdechu rozszerzają się zasysając powietrze przez nos lub usta (lub jednocześnie). W czasie wydechu płuca zaś są ściskane przez odpowiednie mięśnie klatki piersiowej wypychając powietrze zmieszane z dwutlenkiem węgla powstałym w procesie oddychania, w kierunku jamy nosowej i ustnej. W przypadku języka polskiego i znakomitej większości języków świata, fonacja odbywa się podczas wydechu. Oskrzela i tchawica w procesie mowy mają za zadanie doprowadzić strumień powietrza z płuc do krtani. W krtani znajdują się fałdy głosowe, będące podstawowym narządem, służącym wytwarzaniu dźwięcznych fragmentów mowy. Rodzaj fonacji, czyli rodzaj dźwięku, jaki jest tworzony w danym momencie mowy, ściśle zależy od ułożenia strun głosowych. Ich drgania są podstawą dla głosek dźwięcznych, natomiast dla głosek bezdźwięcznych pozostają mniej lub bardziej rozwarte (nie drgają). Podczas normalnego oddychania powierzchnia głośni wynosi u dorosłych około 1 cm2, natomiast podczas fonacji średnio 0,05 do 0,1 cm2. Ciśnienie powietrza pod głośnią jest prawie równe ciśnieniu w płucach. Ciśnienie podgłośniowe stanowi o głośności i wysokości dźwięku. Płuca, oskrzela, tchawica, krtań i struny głosowe można zinterpretować jako źródło sygnału (mowy), swego rodzaju generator wymuszeń. Wszystkie narządy, znajdujące się powyżej wymienionych i wchodzące w skład kanału głosowego, tworzą tak zwane wnęki rezonansowe dodatkowo kształtujące ten sygnał. Wnęki rezonansowe zwane są też niekiedy komorami rezonansowymi lub po prostu rezonatorami. Układ języka, podniebienia, warg, żuchwy, zębów i policzków, wpływa na kształt traktu głosowego. Dodatkowo w przypadku głosek nosowych, zamknięta jama ustna pełni rolę bocznika akustycznego, a dźwięk emitowany jest przez jamę nosową i nozdrza. Jednak rezonanse powstają nie tylko w wymienionych wnękach, ale także w krtani i klatce piersiowej. Mają one jednak znacznie mniejszy wpływ na postać sygnału. Rezonatory pełnią rolę filtra, wzmacniającego jedne, a tłumiącego inne częstotliwości zawarte w sygnale krtaniowym. Harmoniczne wzmocnione noszą nazwę formantów. Są zasadniczą cechą dźwięków mowy. Oznacza to, że każdą samogłoskę i każdą spółgłoskę dźwięczną można opisać za pomocą charakterystycznych tylko dla niej formantów. Tak więc traktując kanał głosowy jako układ

złożony z generatora wymuszeń, filtra (wnęki rezonansowe) o zmiennych parametrach oraz impedancji (usta), wynikowe widmo danej głoski powstaje z nałożenia charakterystyki traktu głosowego, gdzie poszczególne rezonanse są oznaczone w postaci maksimów charakterystyki częstotliwościowej, na widmo tonu krtaniowego. Rezultatem jest sygnał, którego widmo zależne jest od konfiguracji narządów mowy w momencie artykulacji danej głoski. Należy wyraźnie zaznaczyć, iż ton krtaniowy generowany jest w wyniku drgań strun głosowych i jest źródłem dźwięku jedynie dla głosek dźwięcznych. Dla głosek bezdźwięcznych źródłem jest szum – struny głosowe nie drgają i pozostają rozchylone." [6]

III. Testowanie

Testowanie syntezatora mowy różni się od sprawdzania sprawności językowych człowieka. Gdy zostanie wprowadzony do syntezatora tekst "król Karol kupił królowej Karolinie korale koloru koralowego", "stół z powyłamywanymi nogami" lub "w szczebrzeszynie chrząszcz brzmi w trzcinie" nie sprawi to trudności w odczycie, podczas gdy człowiek może mieć z nimi trudności. Działanie oprogramowania przedstawionego w niniejszej pracy (pakiet "SYNT") zależne jest od definiowalnych zasad odczytu. Testy wykonano na przykładowej bazie językowej nazwanej "basia_fonemy.synt".

3.1 Długie ciągi znaków

Testowanie syntezatora może polegać na wprowadzaniu odpowiednio długich ciągów znakowych. Jednym z najdłuższych wyrazów nierozłącznych w języku polskim to "konstantynopolitańczykiewiczówianeczka". Jeżeli po wprowadzeniu złożonych wyrazów odczyt dokonany zostanie poprawnie, można przyjąć, że narzędzie spełnia podstawowe wymogi. [7]

Wyniki : przeprowadzony test zgodnie z wyżej przedstawioną zasadą dla syntezatora "SYNT" wypadł pozytywnie. Zanim otrzymano rezultat należało ustawić prędkość odczytu sampli. Wymieniona wartość została zapisana i nie wymaga ponownej kalibracji.

---

[6] Robert Rodakowski "Opracowanie modułu konkatenacyjnej syntezy mowy polskiej na podstawie
tekstu do zastosowań w komputerowej terapii wad wymowy dzieci" Akademia Górniczo – Hutnicza
im. Stanisława Staszica w Krakowie.

[7] pis.por. http://syntezatorek.republika.pl/

3.2 Nieznane znaki

Kolejnym testem może być umieszczenie wewnątrz wyrazów znaków z poza alfabetu, lub całkowicie nieznanych. Przykładowym ciągiem jest ”wi$$tam@cię”. Dobrze, gdy algorytm dokona interpretacji takiego zbitka znakowego. Znakomicie, jeśli będzie chciał się nauczyć nie rozpoznanych elementów ciągu. [8]

Wyniki : syntezator zauważa obecność niezdefiniowanych wzorców znakowych w tekście. Po odczytaniu znanych fragmentów, podany zostaje komunikat ”Nie wszystko zostało odczytane poprawnie”. Użytkownik może wtedy zdefiniować nieznane wzorce znakowe.

3.3 Prędkość odczytu

Test prędkości odczytu ma bardziej charakter informacyjny o tym, z jaką szybkością syntezator potrafi zrozumiale składać tekst. Szybciej nie znaczy lepiej... Prędkość odczytu tekstu, jaka uważana jest za ”odpowiednią” różni się w zależności od odbiorcy. Jedni preferują wolniejszy odczyt, ponieważ łatwiej się skupiają nad tak dostarczaną treścią, inni natomiast wybiorą szybsze wypowiedzi. [9]

Wyniki : Syntezator nie posiada możliwości regulacji prędkością wymowy. Istnieje natomiast możliwość regulacji długości przerwy pomiędzy odtwarzanymi wzorcami dźwiękowymi (samplami).

3.4 Akcentowanie wyrazów

Akcentowanie wyrazów jak i całych zdań wymaga użycia specjalnego algorytmu. Niewiele syntezatorów go posiada. Forma testowania może wyglądać następująco:
-słowa ”mama”, ”tata”, ”baba”, ”alibaba” posiadają dwie jednakowe sylaby sąsiadujące ze sobą, jednak mimo to sylaby te powinny być czytane w różny sposób (na ogół pierwszy fonem bardziej akcentowany niż drugi) – nie powinny one brzmieć tak samo.
Słowa krótkie typu ”na”, ”do”, ”za” lub ”pod” nie powinny brzmieć jakby były wyjęte z innych wyrazów. [10]

[8] Ibidem
[9] Ibidem
[10] Ibidem

Wyniki : Baza językowa ”basia_fonemy.synt” jakiej używał syntezator podczas testowania, nie akcentowała wymienionych słów. Teoretycznie istnieje możliwość by syntezator pakietu ”SYNT” akcentował odpowiednie fragmenty wyrazów. Można tego dokonać wprowadzając odpowiednie definicje struktur językowych.

3.5 Literowanie skrótów nazw oraz inicjałów

Literowanie skrótów nazw oraz inicjałów. Problem, jaki trzeba brać pod uwagę występuje w brew pozorną dość często w każdym języku. Trudno by program czytał skrót ”WWW” jako niezrozumiały wyraz a nie ”wu wu wu”. Podobnie ”SDK” musi odczytać ”es de ka” a nie ”sydyk” lub w inny sposób. [11]

Wyniki : Otrzymano pozytywne wyniki dla powyżej przedstawionego testu. Syntezator poprawnie wymówił skróty takie jak: ”WWW” , ”SDK” , ”SMS” .

3.6 Odczyt umownych skrótów nazw

Inaczej powinno się odczytywać skróty nazw, a inaczej skróty przyjęte w danym języku. Można odnaleźć około sto dwadzieścia skrótów, jakie powinien znać i poprawnie odczytywać syntezator. Mowa o operacjach zamiany ”tel.” na ”telefon, ”m.n.p.m” na ”metrów nad poziomem morza”. Powinien poprawnie odczytywać tytuły naukowe ”inż.” Jako ”inżynier”, ”mgr” Jako ”magister” i wiele innych. [12]

Wyniki : Syntezator poprawnie odczytuje skróty językowe. Testowana baza językowa nie ma zapisanych ich zbyt wielu, jednak można ją rozbudowywać. Poprawnie odczytane zostały ”inż.”, ”mgr”, ”tel.”.

3.7 Wyjątki językowe

Istnieje cała masa wyjątków, jakie odczytuje się zupełnie inaczej niż zapisuje. Na przykład słowo ”tarzan” nie ma nic wspólnego z ”tarzaniem się” a powinno się raczej tyczyć człowieka zamieszkującego dżungle. Podobnie słowa ”marznąć”, ”zamarznąć”, ”przymarznął”. Problematycznym okazuje się odczyt słowa ”klient”, ”klientowi”, ”kliencki”,

[11] Ibidem
[12] Ibidem

tego typu odmiany syntezator powinien czytać rozdzielając fragment ”kli” od ”ent” lub przedłużając literę ”i”, aby było ją właściwie słychać.

Wyniki : Nie zawsze poprawnie zostają odczytywane wyjątki. Wymagane jest uprzednie zdefiniowanie wyjątków. W bazie ”basia_fonemy.synt” zdefiniowano kilka wyjątków jak np. ”tarzan” i ”marznąć”. Odczyt wymienionych wyjątków został zrealizowany poprawnie.

3.8 Zapis cyfrowy

Kolejnym problemem jest zapis cyfrowy. W europie stosuje się zapis arabski oraz rzymski. Może występować trudność w odróżnieniu cyfr rzymskich od wyrazów, gdyż cyfry rzymskie składają się z liter alfabetu rzymskiego (alfabet łaciński). Jeśli przyjmuje się jedynie obsługę zapisu arabskiego, to nie oznacza to końca występowania błędów przy interpretacji znaków. Na przykład liczba ”1304007”. Przez prosty algorytm zostanie odczytana następująco: ”jeden trzy zero cztery zero zero siedem”. Lepszym wynikiem jest odczyt ”jeden milion trzysta cztery tysiące siedem”. Natomiast, gdy zamiast ”jeden milion” powie ”milion” (pomijając ”jeden”), będzie to oznaczało obecność dobrego algorytmu.

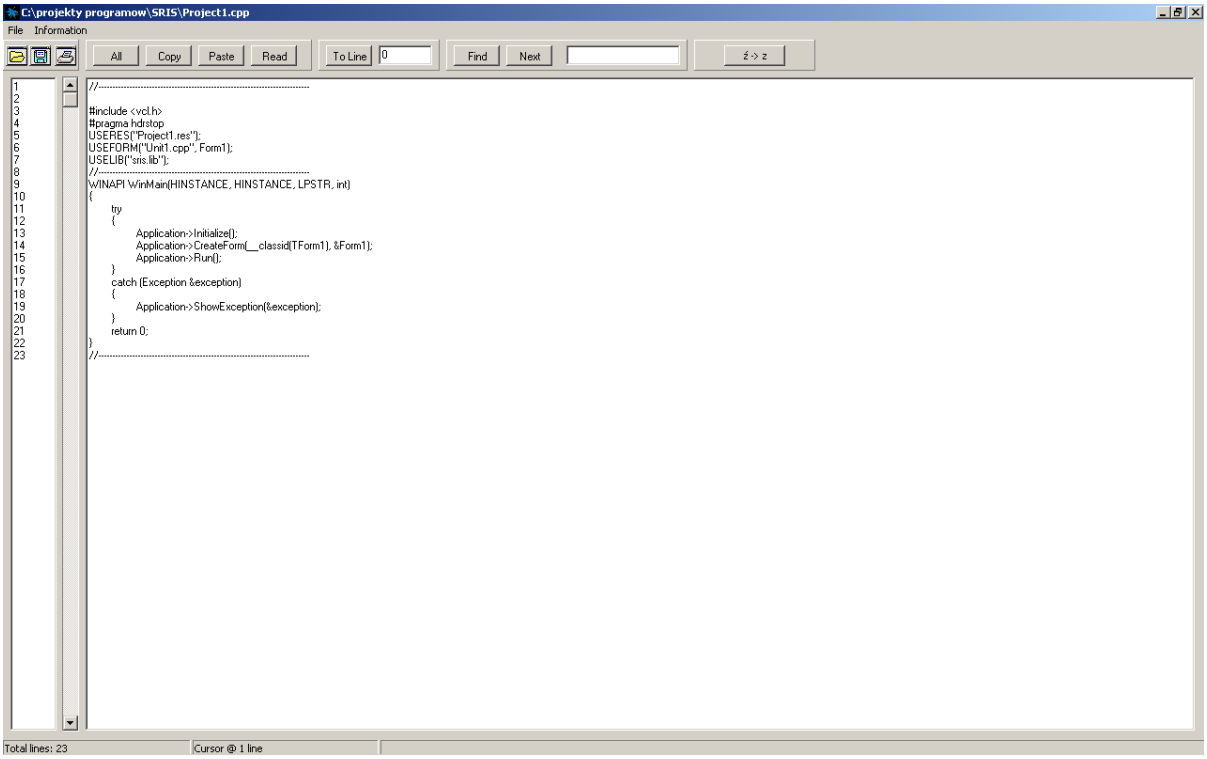
Wyniki : Odczyt zapisu cyfrowego jest realizowany poprawnie za pomocą struktur. Syntezator korzystając z testowanej bazy językowej, potrafi odczytywać liczby całkowite. Po wprowadzeniu ułamków występują błędy w odczycie.

IV. Wdrożenie

Syntezator pakietu ”SYNT” może pracować z dowolnym programem, wystarczy, że programista zamieści kod wysyłający do schowka systemowego tekst do odczytu z dyrektywą poprzedzającą komunikat ”:SYNT: ”. Przykładowy kod procedury w języku C++ wysyłający tekst do syntezatora uruchomionego w tle (syntezator na pasku tray) przedstawiono w ”załączniku 7”.

4.1 Edytor HPC

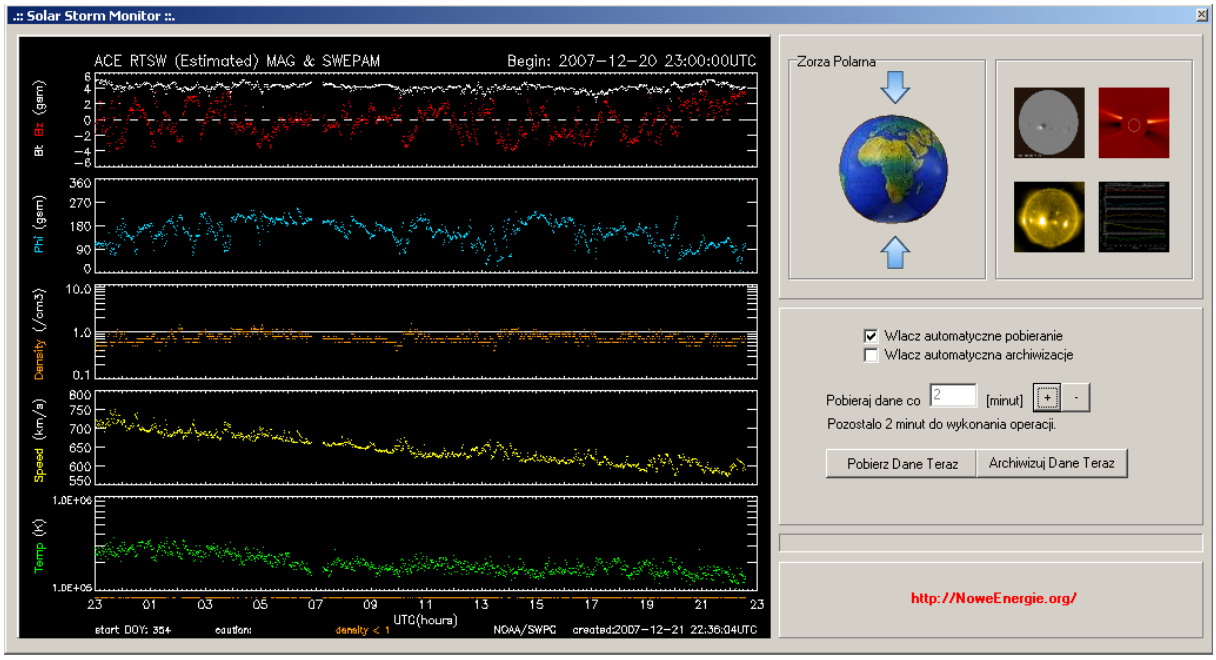
Jednym z programów, jakie zostały podłączone do syntezator pakietu ”SYNT” jest ”Edytor HPC” przedstawiony na Rys. 4.1. Program jest edytorem tekstu, kodu, obsługuje pliki formatu *.txt *.php *.php3 *.inf *.css *.wml *.htaccess . Jest on przeznaczony dla programistów, pracujących w czystym kodzie. Można w nim edytować, wczytywać, drukować, zapisywać, wyszukiwać w tekście, przenosić się do wskazanej linii kodu, oraz wycinać ”cut”, wklejać ”paste”, kopiować ”copy” łańcuchy znaków. Rozwiązaniem, nie spotykanym w innych programach tego typu jest zamiana polskich znaków takich jak ”ą” ”ę” na ”a” , ”e”. Odczyt tekstu wczytanego do edytora następuje po kliknięciu przycisku ”Read”. Program został opublikowany w Internecie w bibliotekach oprogramowania.



Rys. 4.1 Screen programu ”Edytor HPC”

4.2 Solar Storm Monitor

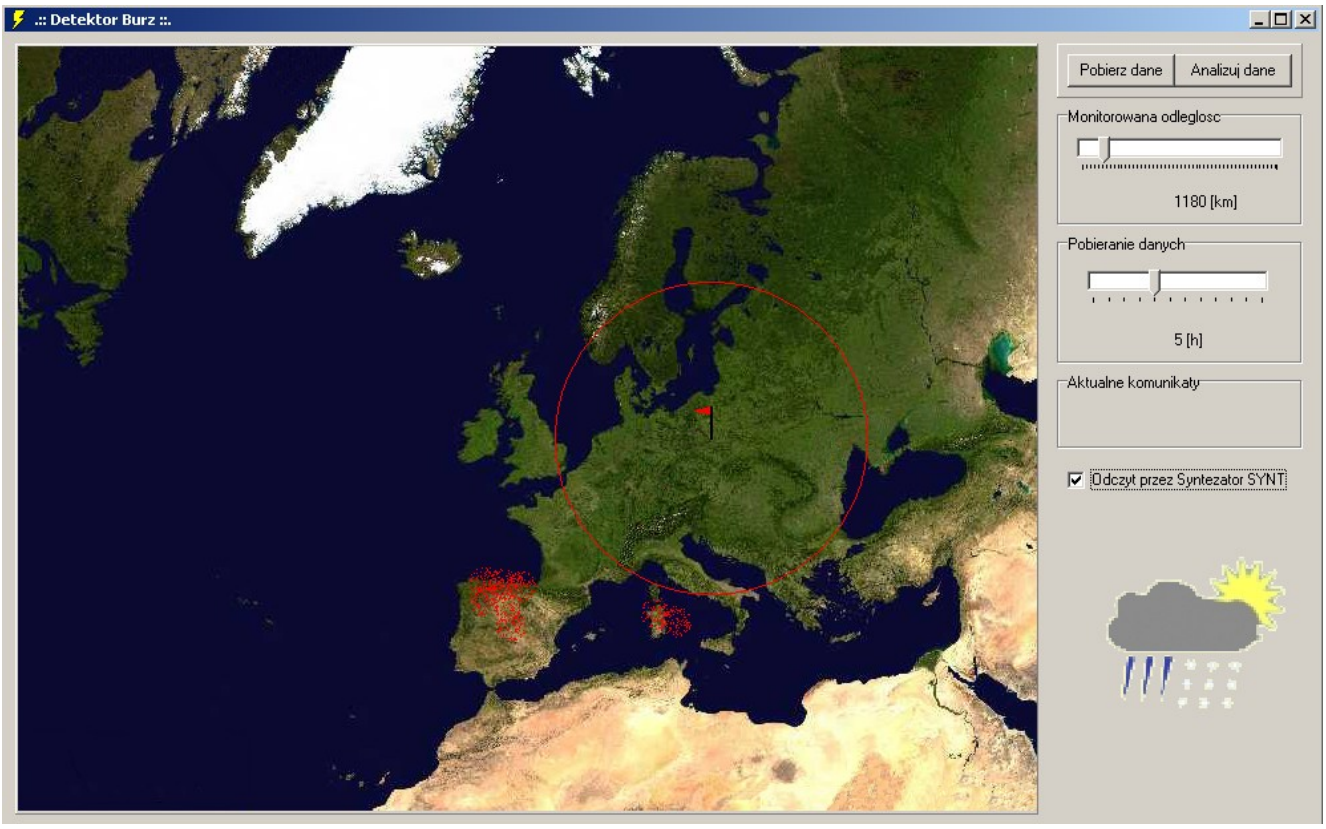
Kolejnym programem, w jakim znalazł zastosowanie syntezator mowy pakietu ”SYNT” jest ”Solar Storm Monitor” przedstawiony na Rys. 4.2. Jest to aplikacja pobierająca dane przez Internet z satelity SOHO, jaki krążąc na orbicie okołoziemskiej monitoruje aktywność słoneczną oraz poziom wysokoenergetycznych cząstek mających wpływ na powstawanie zorzy polarnej. Praktyczne zastosowanie owego programu jest bardzo szerokie, ponieważ wybuchy na słońcu mają wpływ nie tylko na powstanie zorzy polarnej, ale także na łączność radiową. Dane pobrane z satelity SOHO są analizowane i przedstawiane w postaci wykresów, oraz w postaci graficznej przedstawiającej aktualny obraz zorzy polarnej a także protuberancji słonecznych. W przypadku powstania burzy słonecznej syntezator otrzymuje stosowny komunikat i wypowiada go, ostrzegając o zaistniałej sytuacji. Burze słoneczne są bardzo niebezpieczne dla sprzętu elektronicznego, w szczególności dla satelit krążących na orbicie okołoziemskiej, a także dla systemów komunikacyjnych i energetycznych. Znane są przypadki zniszczenia sprzętu elektronicznego w czasie burzy słonecznej.



Rys. 4.2 Screen programu ”Solar Storm Monitor”

4.3 Detektor burz

Ostatnim programem, który został związany z syntezatorem mowy jest detektor burz przedstawiony na Rys.4.3. Pobiera on z Internetu informacje na temat aktualnych wyładowań atmosferycznych i przedstawia je na mapie w formie pojedynczych czerwonych punktów. Klikając na mapie zostaje ustalona pozycja bazowa (czerwona chorągiewka), wokół której tworzy się obszar monitorowania (czerwony okrąg). Jeśli w owym obszarze pojawią się wyładowania, program zmierzy, w jakiej odległości one wystąpiły i zawiadomi o tym poprzez syntezator oraz w formie komunikatu głosowego.



Rys. 4.3 Detektor Burz

Zakończenie

Wnioski

Zaawansowane programy TTS obecne na rynku są niezmiernie drogie, trudno powiedzieć czy powodem ich ceny jest algorytm (często opracowywany od podstaw), czy też prawa rynku. Dziedziny, w których syntezatory mowy mogą znaleźć zastosowanie jest niezmiernie wiele. Wciąż jednak stosuje się mniej zaawansowane suplementy. Wiele aplikacji posiada własny syntezator (np. program nawigacyjny) zamiast używać syntezatora, który mógłby zostać wbudowany w system operacyjny i potrafiłby obsługiwać wszystkie aplikacje. Wiele systemów ”Call Center” wciąż nie posiada syntezatora a jedynie suplement. Idea stosowania suplementu polega na tym, ze zamiast jednego zaawansowanego syntezatora TTS występuje stosunkowo prosty algorytm operujący paroma wyrazami, jakie w połączeniu w całość tworzą werbalny komunikat. Wymowa jest bardzo dobra, jednak nie ma możliwości odczytu dowolnego tekstu. Prezentowane rozwiązanie w tej pracy daje możliwość budowy zarówno prostych systemów, jakie operują zaledwie paroma wyrazami (np. ”skręć w prawo” , ”skręć w lewo”), oraz umożliwia budowę syntezatora, który czyta tekst dowolny. Można też stosować wszelkiego rodzaju mutacje typu: zaawansowany odczyt wybranych wyrazów, z uwzględnieniem odczytu dowolnego tekstu w gorszej jakości, i wiele innych. Zaletą i nowością prezentowanego tu pakietu ”SYNT” jest fakt, że można zbudować dowolny język w dowolnej chwili bez potrzeby zmiany oprogramowani, ponownej kompilacji, ingerencji w algorytm TTS. Wszelkie potrzebne modyfikacje wykonuje się manipulacją bazą danych struktur i sampli (alfabetu). W trakcie testów został zauważony problem występujący przy analizie struktur, nie wpływa ona na poprawność działania programu, ale na ekonomię odczytu dłuższych tekstów. Testowano syntezator zarówno przy tekstach typu ”komunikat” (do 200 znaków), oraz w przypadkach tekstów typu ”książka” (wiele tysięcy znaków), które wykazały zastosowanie szybkiego procesora i dysku twardego o dobrych parametrach czasu dostępu. Szacunkowo w języku polskim wyrazy posiadają długość około ośmiu znaków. Zakładając, że baza struktur zawiera sto rekordów z hasłem średnio jedno znakowym to przy długości tekstu wynoszącym tysiąc znaków będzie potrzeba około stu tysięcy operacji by tylko zlokalizować struktury w tekście. Rozpoznanie samej struktury jest zależne od jej długości i wymaga wykonania kolejnych pętli analizujących. Średnio by odczytać tekst o długości tysiąca znaków i przy zdefiniowanych stu strukturach trzeba wykonać kilkaset

tysięcy operacji. Algorytm analizy struktur znacznie wpływa na czas potrzebny do rozpoczęcia odczytu tekstu.

Perspektywy rozwoju

Pakiet ”SYNT” wymaga opracowania bardziej zaawansowanych baz sampli, oraz struktur językowych. Tyczy się to zarówno zastosowań dedykowanych (”call center”, systemy alarmowe) jak i ogólnych (odczyt dowolnych tekstów). Stworzenie aplikacji działającej po stronie serwera internetowego, w połączeniu np. ze skryptami JAVA umożliwiłoby odczyt treści witryny internetowej po stronie klienta. Umożliwi to łatwiejszy dostęp do strony nie tylko osobą niewidomym, można taki zabieg potraktować również jako ciekawy chwyt marketingowy. Innym rozwiązaniem odczytującym treść witryn internetowych, jest stworzenie tak zwanego ”screen reader’a”. Ponieważ syntezator może odczytywać książki, nie trzeba pobierać audio book’ów, wystarczy wczytać tekst do syntezatora, by pełnił on funkcję lektora. Rozszerzenie tej opcji może odczytywać tekst do pliku, zapisując go w formacie *.mp3. Umożliwiłoby to przechowywanie odczytanego tekstu w przenośnym odtwarzaczu. Syntezator może być przydatny w edukacji. Wykorzystywać go można nie tylko do czytania bajek dzieciom, ale i w trakcie nauki języka. Przydatną opcją przy wymienionym zastosowaniu jest podświetlanie czytanego tekstu. Natomiast odtwarzacz filmów współpracujący z syntezatorem pozwoli skupić się na filmie, a nie śledzić ciągnące dialogi. Specjalny program, lub nawet sam syntezator może służyć nauce tekstów, na przykład wiersza, lub do odsłuchiwania własnego przemówienia. Opracowanie programu do odczytu wiadomości z kanałów RSS, umożliwiłoby użytkownikowi komputera podłączonego do Internetu wysłuchiwanie nowości ze świata. Stworzenie aplikacji terminarza, jaka informowałaby o nadchodzących spotkaniach, lub innych wydarzeniach.

Bibliografia

Prace dyplomowe

[D1] Rodakowski Robert :*”Opracowanie modułu konkatenacyjnej syntezy mowy polskiej na podstawie tekstu do zastosowań w komputerowej terapii wad wymowy dzieci*”, prom, 2007 Akademia Górniczo – Hutnicza im. Stanisława Staszica w Krakowie.

Strony internetowe

[I1] http://pl.wikipedia.org/wiki/Syntezator_mowy

[I2] http://pl.wikipedia.org/wiki/Ivo_Software

[I3] http://syntezatorek.republika.pl/

[I4] http://www.users.pjwstk.edu.pl/~kszklanny/syntezam.htm

[I5] http://www.phon.ucl.ac.uk/

Artykuły

[A1] Wojciechowski Adam *”Sterowanie mową w wirtualnej galerii sztuki*”, Instytut Informatyki Politechniki Łódzkiej, październik 2002.

Publikacje książkowe

[K1] Kent Reisdorph *”C++ Builder 6”* , HELION, 2003.

[K2] Rob Polevoi *”3D Studio MAX R3 in Depth*”, HELION, 2007.

[K3] Wierzchowska B. *”Fonetyka i fonologia języka polskiego”*, OSSOLINEUM, 1980.

Spis rysunków

Spis załączników

Załączniki

Załącznik 1. “Alfabet”

Funkcja tworząca alfabet na podstawie wprowadzonego tekstu.

```
//pobierz tekst
text=Budowacz->memo_tekst->Text;
//na początek tekstu
int kareta=1;
//długość tekstu
int total_znakow=text.Length(); // -2 jeśli zakończony enterem
//ustalanie długości paska postępu
Budowacz->Progress->Max=total_znakow;
//pasek postępu wyzerować
Budowacz->Progress->Position=0;
while (kareta<total_znakow)
 {
 //jeśli znak jest z poza alfabetu i nie jest spacją
 if (czy_jest_w_alfabecie(get_znak(kareta))==false && get_znak(kareta)!=’ ‘)
     {
     //dodaj do alfabetu
     Budowacz->alfabet->Items->Add(get_znak(kareta));
     }
 kareta++;
  //postęp na pasku
 Budowacz->Progress->Position=kareta;
 }
aktualizuj_napisy();
Budowacz->Progress->Position=0;
```

Załącznik 2. “Difony i fonemy”

Funkcja tworząca fonemy, oraz difony na podstawie analizowanego tekstu.

```
//pobierz tekst
tekst=Budowacz->memo_tekst->Text;
//na początek tekstu
int kareta=1;
//długość tekstu
int total_znakow=tekst.Length(); //-2 jeśli kończy się enterem
//ustalanie długości paska postępu
Budowacz->Progress->Max=total_znakow;
//pasek postępu wyzerować
Budowacz->Progress->Position=0;
while (kareta<total_znakow)
 {
 // jeśli znak nie jest w alfabecie i znak nie jest spacją
 if (get_znak(kareta)!=’ ‘ && get_znak(kareta+1)!=’ ‘)
    {
if (czy_jest_w_alfabecie(get_znak(kareta)) && czy_jest_w_alfabecie(get_znak(kareta+1)))
        {
         //pobranie sąsiedniego znaku
         AnsiString dwuznak=get_znak(kareta)+get_znak(kareta+1);
         kareta++;
         //jeśli nie ma go w bazie
         if (czy_jest_w_bazie_powiazan(dwuznak)==false)
            {
            //dodaj do bazy
            Budowacz->powiazania->Items->Add(dwuznak);
            }
         }
    }
 //następny znak
 kareta++;
  //postęp na pasku
 Budowacz->Progress->Position=kareta;
 }
aktualizuj_napisy();
Budowacz->Progress->Position=0;
}
 else
     {
     ShowMessage(„Najpierw dokonaj analizy wyciągającej alfabet z tekstu \n Następnie
go   przeanalizuj usuwając niepotrzebne znaki \n potem można opracować powiązania”);
 }
```

Załącznik 3. ”Funkcje uzupełniające”

Funkcje używane w algorytmach przedstawionych w ”załączniku 1” oraz ”załączniku 2”.

```
//-------------------------szukaj znaku w alfabecie-----------------------------
bool czy_jest_w_alfabecie(AnsiString sampel)
{ int index=Budowacz->alfabet->Items->IndexOf(sampel);
   if (index>-1)
    { return true; }
                    else
                       { return false; }}
//-------------------------zwraca true jeśli ciąg jest w bazie sampli-------------
bool czy_jest_w_bazie_powiazan(AnsiString sampel)
{ int index=Budowacz->powiazania->Items->IndexOf(sampel);
   if (index>-1)
    { return true; }
                    else
                       { return false; }}
//-------------------------pobierz znak z tekstu jako char--------------------------
AnsiString get_znak(int pozycja)
{ //zamień wszystko na małe znaki
  //zamiana znaku na ascii
  int ascii_znak=tolower(tekst[pozycja]);
  //ascii do char
  AnsiString znak=(char)ascii_znak;
  return znak; }
```

Załącznik 4. “Struktura”

Budowa struktury oraz definicja tablicy struktur.

```
//---------------------Definicja struktury-----------------------------------
struct klucz
{
bool przed;
bool po;
char maska[4];
char haslo[20];
char efektor[30];
};
//---------------------------------------------------------------------------
klucz Strukt[99];  // 0 to 99 = 100
```

Załącznik 5. “Odczyt”

Algorytm odczytu tekstu.

```
//zmienna z czasem pomiędzy odczytem sampli
int o_speed=speed;
//zmienna ze ścieżką do sampli
AnsiString sciezka;
//jeśli ciąg posiada znaki
if (total_znakow!=0)
{//pozycja gdzie zakończono odczyt
int kareta_lokalna=kareta;
//zmienna z tekstem do szukania najdłuższego sampla
AnsiString bufor="";
//zmienna z tekstem posiadającym odpowiednik w bazie sampli
AnsiString sampel="";
//gdy nie spacja to słowo
if (get_znak(kareta_lokalna)!=' ')
{//szukanie najdłuższego sampla
while (get_znak(kareta_lokalna)!=' ')
     {//gdy nie jest koniec tekstu
     if (total_znakow!=kareta_lokalna)
          {//dodaj znak do bufora
          bufor=bufor+get_znak(kareta_lokalna);
          //bufor jest najdłuższym samplem w bazie
          if (sprawdz_sampel(bufor)){
             o_speed=speed+bufor.Length()*20;
             //bufor jest samplem
             sampel=bufor;
             //ścieżka do sampla
             sciezka=path_sampli+sampel+".wav";
             //zmiana pozycji w tekście
             kareta=kareta_lokalna+1;
             }
          //następny znak
          kareta_lokalna++;
          }
       }
// sampl nie istnieje
if ( sampel=="") {kareta=kareta_lokalna;
                  bledy_odczytu=true;
                  //ShowMessage("Nie potrafię odczytać "+bufor);
                 }
                 //jeśli istnieje czytaj go
                 else
                     {//zamien ansistring na char
                     char *path=new char[ sciezka.Length()+1 ];
                     strcpy( path, sciezka.c_str() );
                     //wysłanie do karty dźwiękowej
                     sndPlaySound(path, SND_ASYNC);
                     }
}
//znak jest spacją
else         {//czekaj
               o_speed=speed+200;
               //do następnego znaku w tekście
               kareta++;
              }
}
 //przerwa między samplami
Syntezator->Timer1->Interval=o_speed;
 //---koniec odczytu
 if (total_znakow<=kareta+1)
      {Syntezator->Timer1->Enabled=false;
       if (bledy_odczytu==true)
          {
           String bufor=":SYNT:Nie wszystko odczytane poprawnie";
           pCB->AsText=bufor;
```

```
        }
     }
```

Załącznik 6. “Test schowka”

Fragment kodu sprawdzającego czy w schowku znajduje się komunikat dla syntezatora.

```
if (pCB->HasFormat(CF_OEMTEXT))
  {
  String bufor= pCB->AsText;
  if (bufor.Pos(":SYNT:"))
     {
     bufor=bufor.Delete(1,6);
     Syntezator->memo_tekst->Clear();
     Syntezator->memo_tekst->Text=bufor+" ";

     start_czytania();
     pCB->Clear();
     }
  }
```

Załącznik 7. ”Komunikat”

Przykład procedury wysyłającej komunikat do odczytu przez syntezator.

```
void  odczytaj(String tekst)
{
#include <clipbrd.hpp> //obsługa schowka windows
Tclipboard *Pcb = Clipboard();
String bufor=”:SYNT: „+tekst;
Pcb->AsText=bufor;
}
```