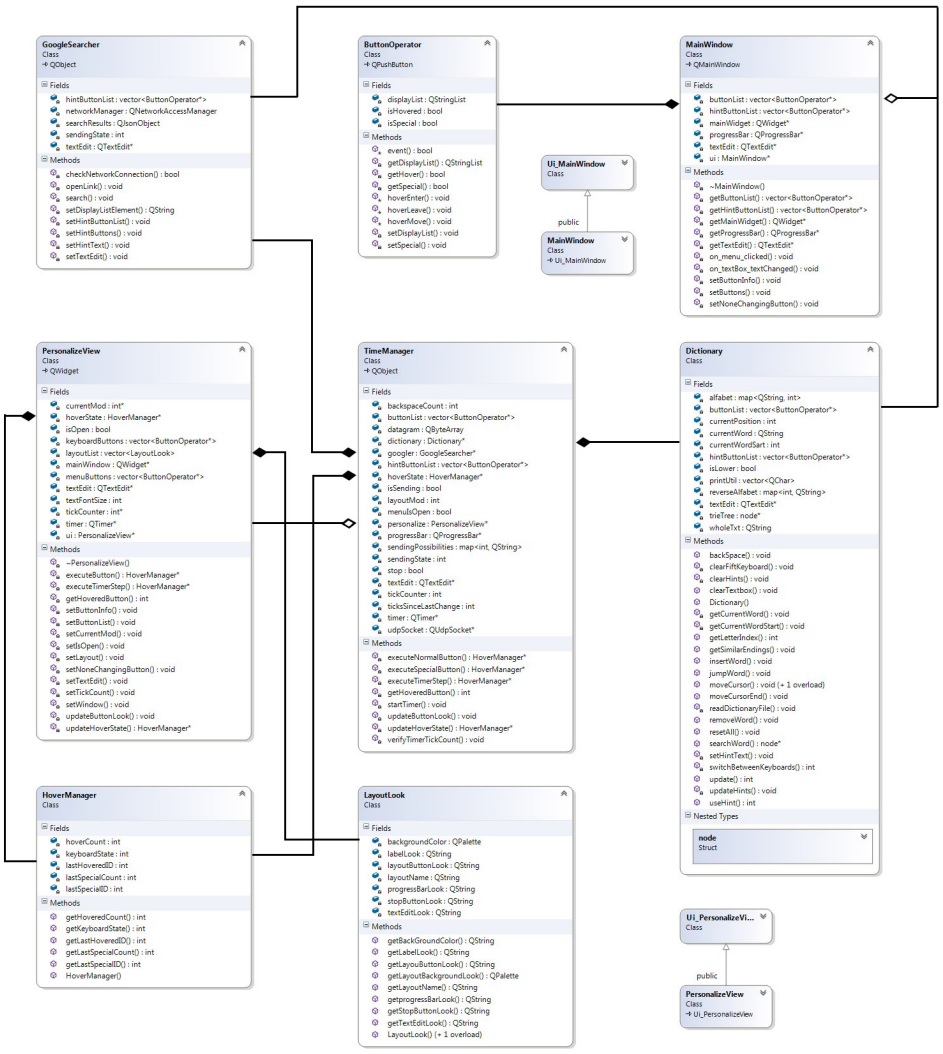
Rozdział 1

Implementacja

W aplikacji, wykorzystującej wyżej wymienione technologie, stworzono: 8 klas w języku C++ wraz z odpowiadającym im plikom nagłówkowym, dodatkowy plik zawierający stałe programowe oraz 2 widoki.

1.1 Diagram klas

Na rysunku 1.1 przedstawiono poglądowy diagram klas wraz z ich zależnościami. Opis klas oraz zastosowanych algorytmów zawarto w podrodziałach 1.2 oraz 1.3.



Rys. 1.1 – Poglądowy diagram klas zaimplementowanych w projekcie.

1.2 Opis zastosowanych klas

Klasami użytymi w programie są: ButtonOperator, Dictionary, GoogleSearcher, HoverManager, LayoutLook, MainWindow, PersonalizeView, TimeManager. Dla każdej z nich można określić główną funkcję. Zestawienie to przedstawiono w tabeli 1.1.

Nazwa klasy	Funkcja w programie
ButtonOperator	Klasa dziedzicząca po QPushButton (wewnętrzna klasa biblioteki QT) odpowiedzialna za działanie przycisków w projekcie.
Dictionary	Klasa zajmująca się pracą ze słownikiem, a także zarządzaniem wprowadzonym tekstem. Kontruktor przyjmuje wskaźnik na pole tekstowe, listę wskaźników na wszystkie działające przyciski oraz listę wskaźników na przyciski odpowiedzialne za podpowiedzi. Klasa posiada również strukturę "node", która stanowi podstawowy element budulcowy dla słownika. Więcej na ten temat w podrozdziale 1.3.3.
GoogleSearcher	Klasa zajmująca się komunikacją programu z internetem, obsługą żądania GET oraz przetworzeniem odpowiedzi. Więcej na ten temat w podrozdziale 1.3.5.
HoverManager	Klasa nadzorująca działanie przycisków. Obiekty przechowują informację o: ID przycisku ostatnio fiksowanego, ID ostatnio wybranego specjalnego przycisku, czasie fiksacji wzorku na jednym przycisku, czasie działania specjalnego przycisku. Więcej o przyciskach i ich działaniu w podrozdziale 1.3.1.
LayoutLook	Klasa przewidziana z myślą o dynamicznej zmianie wyglądu aplikacji przez użytkownika w oknie menu. Obiekty klasy zebrane w listę w klasie PersonalizeView przechowują komplet danych dotyczących jednego wyglądu. W skład tego wchodzą: nazwa wyglądu, wygląd przycisków zapisany przy pomocy kaskadowego arkusza styli, paletę zmieniającą kolor tła okien, arkusz styli dla paska postępu, podpisy oraz wygląd przycisków w trybie zablokowanym.
MainWindow	Gówna klasa projektu odpowiedzialna za komunikację interfejsu z resztą klas.
PersonalizeView	Klasa odpowiedzialna za obsługę wszystkich zdarzeń związanych z oknem menu, tzn. zmianą wyglądu aplikacji, zmianą wielkości czcionki, zmianą czasu progowego fiksacji.
TimeManager	Rozbudowana klasa będąca głównym zarządcą procesów zachodzących w aplikacji dzięki umiejętności pracy z regulatorem czasowym (timer).

Tab. 1.1 – Tabela przyporządkowania głównych funkji klasom projektu.

1.3 Zaimplementowane algorytmy

W poniższych podrozdziałach zaprezentowano zasadę działania zaimplementowanych w oprogramowaniu algorytmów.

1.3.1 Działanie przycisków

W aplikacji każdy przycisk jest obiektem klasy ButtonOperator, która dziedziczy po klasie QPushButton. Dzięki czemu obiekty przycisków mogą korzystać zarówno z metod klasy nadrzędnej np. isCheckable(), jak i klasy dziedziczącej. Właśnie dzięki nadpisaniu metod domyślnych klasy QPushButton - hoverEnter() i hoverLeave() stworzono możliwość detekcji, nad którym przyciskiem aktualnie znajduje się punkt fiksacji wzroku. Podczas wywołania obu metod zmieniana jest wartość logiczna zmiennej isHovered aktualnie obserwowanego przycisku. Prócz informacji o tym, czy dany przycisk jest aktualnie używany przechowuje się również dane o tym, czy przycisk zalicza się do tzw. "specjalnych", czy też nie, jak i listę dostępnych dla niego tekstów do wyświetlania – wyglądów przycisku dla 6 stanów klawiatury (małe litery, duże litery, znaki specjalne karta pierwsza, znaki specjalne karta druga, polskie litery, menu kontekstowe). Wszystkie te dane wprowadza się podczas uruchamiania klawiatury i wtedy także przypisuje się przyciski kolejno do specjalnej listy. Kolejność jest znacząca w przypadku przycisków "specjalnych", gdyż ich obsługa zależna jest od wartości ich ID zapisanego w pliku ze stałymi. Określenie przycisku działającego odbywa się poprzez sprawdzenie jego ID, którego zmienna isHovered jest prawdziwa w klasie TimeManager. Do zarządzania stanem klawiatury, ostatnio wybranym przyciskiem specjalnym oraz ostatnio obserwowanym przyciskiem powstała klasa HoverManager. Zbiera ona na bieżąco informację o ID przycisku ostatnio najechanego, o tym przez jaki czas dany przycisk jest już pod punktem fiksacji, o aktualnym stanie klawiatury, o ostatnio wybranym specjalnym przycisku (np. CapsLock) oraz przez jaki czas działanie tego przycisku się utrzymuje. Większość z tych danych odświeżana jest co 200ms (stała zdefiniowana w pliku ze stałymi) podczas każdego wykonywania się metody TimerStep(). Jej działanie przedstawiono za pomocą pseudokodu 1.
Zadaniem wyżej wspomnianej funkcji executeTimerStep() jest sprawdzenie, czy dany przycisk był fiksowany przez odpowiednią ilość czas (zmienna, której wartość zależy od ilości pomyłek popełnionych przez użytkownika, jest dynamicznie zmieniana podczas korzystania z klawiatury – analizowane to jest w funkcji verifyTimerTickCount()). Jeżeli warunek ten został spełniony to dalszy przebieg działań zależy od tego, czy przycisk był specjalny, czy też nie oraz czy klawiatura nie znajduje się w trybie wstrzymania.

1.3.1.1 Działanie przycisków specjalnych

W tabeli 1.2 wymieniono wszystkie specjalne przyciski z widoku klawiatury oraz opisano po krótce algorytm ich działania. W tabeli 1.3 przedstawiono przyciski widoku menu oraz ich zachowanie.

Jako wciśnięcie rozumie się tu czas fiksacji nad przyciskiem przekracający progową wartość. Stany klawiatury to odpowiednio 0-małe litery, 1-wielkie

Nazwa przycisku	Działanie
Backspace	Wywołuję metodą backspace() klasy Dictionary, której działanie opisano w podrozdziale 1.3.2.1.
CapsLock	W przypadku pierwszego wciśnięcia zmienia stan klawiatury z 0 na 1 i utrzymuje ją dopóki nie zostanie wybrany po raz wtóry lub wykonana zostanie czynność innego przycisku zmieniającego stan klawiatury.
Czyść	Korzysta z metody resetAll() klasy Dictionary i powoduje czyszczenie edytora tekstowego oraz z nim związanych zmiennych jak currentPosition, currentWord, currentWordSart, wholeTxt (więcej na ten temat w rozdziale 1.3.2).
Menu	Otwiera okno klasy Personalize.
Przesuń się o jedno słów w prawo lub w lewo	Wywołuje funkcje jumpWord() klasy Dictionary, która powoduje przesunięcie się kursora na koniec poprzedniego słowa, lub na początek kolejnego. Każdorazowo zmieniana jest wartość zmiennej currentWord i wholeText. 1.3.2
Przesuń się w kierunku początku tekstu (home) lub na jego koniec (end)	Wybranie przycisku powoduje przesunięcie się kursora w jednym z dwóch kierunków oraz zmiane wartości zmiennej wholeText i currentWord (1.3.2).
Przyciski podpowiedzi	Wyświetlane są na nich podpowiedzi, których tworzenie opisane jest w rodziale 1.3.3. Ich użycie powoduje zastąpienie aktualnie wpisywanej frazy autouzupełnionym słowem pobranym ze słownika.
Shift	W przypadku wciśnięcia zmieniony zostaje stan klawiatury z 0 na 1, a po użyciu przycisku ze znakiem stan klawiatury wraca do stanu 0.
Stop i start	Wprowadza klawiaturę w stan wstrzymania lub w stan pracy. W zależności od wartości zmiennej isStop jest możliwe, lub nie, korzystanie z przycisków ze znakami.
Strzałka w lewo i prawo	Korzystając z metody moveCursor klasy Dictionary przemieszcza kursor o jeden znak w danym kierunku. Może zmienić wartość currentWord (1.3.2) korzystając z metod getCurrentWord() oraz getCurrentWordStart()(2) opisanych w późniejszych rozdziałach.
Wyjdź	Powoduje opuszczenie aplikacji.
Wyszukaj na podanej platformie	Przyciśniecie przycisku powoduje sprawdzenie połączenia internetowego, a następnie wysłanie wpisanej treści pola tekstowego do funkcji search() klasy GoogleSearcher. 1.3.5
Wyślij	Wysyła dotychczas wpisany tekst na broadcast na port 45454 za pomocą metody writeDatagram() klasy QUdpSocket.
Zmiana trybu wyszukiwania	Strzałki powodują na zmianę trybu wyszukiwania. Do wyboru ”Google”, ”YouTube”, ”Filmweb”. Zmieniają wartość zmiennej sendingState niezbędnej do popranego działania funkcji search() klasy GoogleSearcher. 1.3.5
Znaki specjalne	Zmienia stan klawiatury na odpowiednio 2 i 3 przy kolejnych kliknięciach.

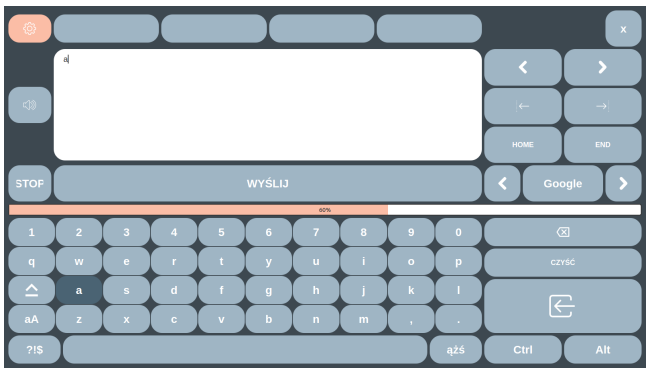
Tab. 1.2 – Lista specjalnych przycisków wraz z ich działaniem.

Algorithm 1 Działanie funkcji TimerStep()

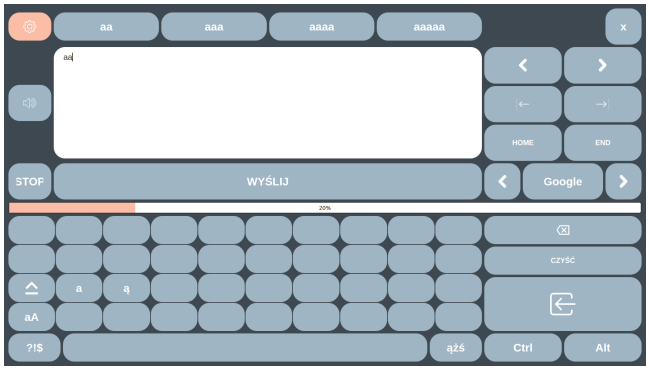
pobierz ID aktualnie fiksowanego przycisku
if pobrane ID różne jest od -1 **then**
  if zwykłe przyciski są w trybie wstrzymania **then**
    if jeśli aktualnie fiksowany przycisk jest specjalny **then**
      hoverState (obiekt klasy HoverManager) ustaw aktualnie aktywny przycisk na pobrane ID
      Wykonaj krok timera (funkcja executeTimerStep())
      Pokaż czas przez jaki przycisk jest fiksowany na pasku postępu dla informacji użytkownika.
    end if
  else
    Wykonaj powyższe funkcje dla wszystkich przycisków - niezależnie od tego, czy są specjalne, czy nie.
  end if
end if
Wywołaj funkcję verifyTimerTickCount().

Nazwa przycisku	**Działanie**
Strzałki zmieniające aktualną wartość progową czasu fiksacji	Strzałki powodują zmniejszenie lub zwiększenie wartości zmiennej tickCounter przechowującej ilość 200ms interwałów, po upływie których (w przypadku braku zmiany fiksowanego punktu) przycisk można uznać za wciśnięty. Początkowo czas ten wynosi 3s, jednak w wyniku dynamicznej korekty następuje zmniejszenie lub zwiększenie liczby tych interwałów. Korekta uruchamia się, gdy użytkownik przekroczy dopuszczalną ilość błędów w ciągu minuty. Więcej na temat dynamicznej zmiany czasu w podrozdziale 1.3.4.
Strzałki zmieniające wielkość czcionki w edytorze tekstu	Strzałki powodują zamianę właściwości edytora tekstu.
Strzałki zmieniające wygląd aplikacji	Użytkownik może wybrać między jednym z 5 wersji kolorystycznych poprzez dynamiczną zmianę wyglądu na podstawie danych z listy i aktualnego indeksu.
Wyjście	Powoduje zamknięcie okna menu.

Tab. 1.3 – Lista specjalnych przycisków z menu w raz z ich działaniem.



Rys. 1.2 – Widok aplikacji z klawiatura w stanie zerowym po wpisaniu litery 'a'.



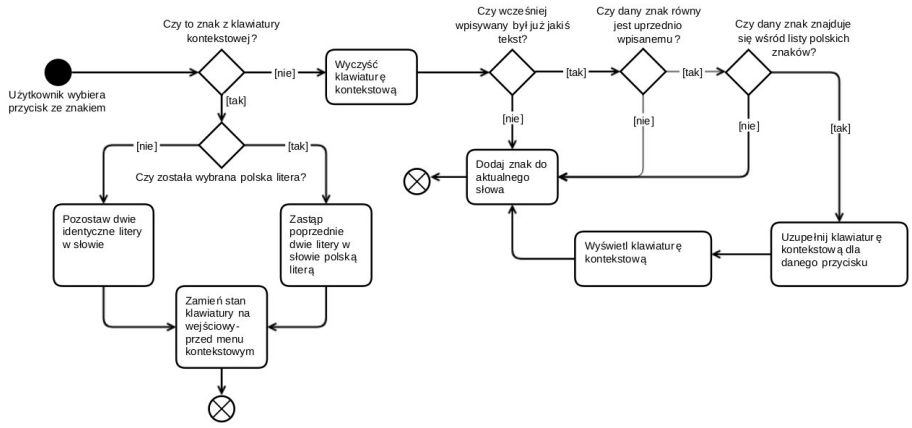
Rys. 1.3 – Widok aplikacji z klawiaturą w stanie piątym po wpisaniu drugiej litery 'a'.

litery, 2-znaki specjalne strona pierwsza, 3-zanki specjalne strona druga, 4-polskie znaki, 5-menu kontekstowe dla polskich znaków.

1.3.1.2 Działanie przycisków nomalnych

Za wpisywanie znaków z klawiatury do edytora odpowiedzialna jest funkcja - update() klasy Dictionary. Jest ona głównym zarządcą jeśli chodzi o wprowadzanie znaków w odpowiedniej pozycji. W pierwszej kolejności sprawdzane jest, czy stan klawiatury różny jest od stanu piątego (menu kontekstowe). Gdy spełniony jest dany warunek to porównywana jest aktualnie wprowadzana wartość z uprzednio wprowadzoną. Wyjątek stanowi pierwsze wprowadzenie znaku do edytora. W przypadku dwóch identycznych liter czyszczona jest zawartość piątego stanu klawiatury i przechodzi się do wywołania funkcji switchBetweenKeyboards(), która zwraca informację o tym, czy stan klawiatury się zmienił. Tam sprawdzane jest, czy wprowadzony znak jest jedną z liter posiadających polskie odpowiedniki tj. *a-ą, c-ć, e-ę, l-ł, n-ń, o-ó, s-ś, z-ż-ź*. Gdy znajdzie się on wśród wymienionej listy, to dla odpowiadającego przycisku ustawiany jest test dla klawiatury stanu piątego i następuje jej wyświetlenie. Działanie takiego zachowania widać na rysunkach 1.2, 1.3.

W innym wypadku aktualna litera dopisywana jest do aktualnie tworzonoego słowa (currentWord), kursor przesuwany jest o jedną pozycję w prawo. Gdy dodana jest spacja aktualnie przetwarzane słowo jest uznawane za skończone i zapamiętywany jest nowy początek kolejnego słowa. Odświeżony zostaje również widok podpowiedzi, których powstawanie omówiono w późniejszym rozdziale 1.3.3. W sytuacji, gdy wybrana zostaje litera z menu kontekstowego (klawiatury w stanie piątym), to albo wpisany tekst zostaje podmieniony na polską literę,lub nie ulega zmianie. Przykładowo po wpisaniu ”aa” użytkownik po raz kolejny wybiera literę ”a” - tzn. planował wpisanie frazy ”aa”. Jeśli jednak decyduje się na literę ”ą”, to w miejsce widniejącego napisu ”aa” pojawia się ”ą”. Poglądowy diagram przepływu przedstawiono no rysunku 1.4.

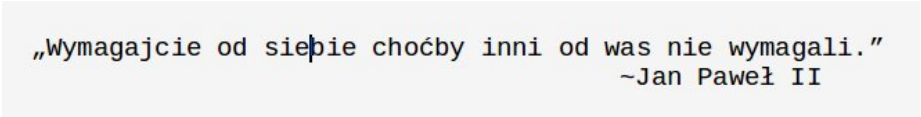


Rys. 1.4 – Widok aplikacji z klawiaturą w stanie piątym po wpisaniu drugiej litery ’a’.

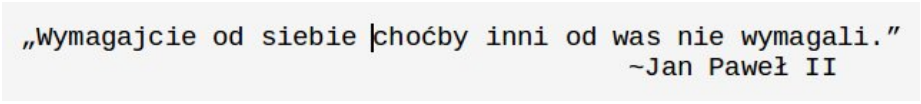
1.3.2 Praca z tekstem

Wpisywanie tekstu w aplikacji odbywa się poprzez specjalny algorytm kontrolujący zawartość aktualnie pisanego słowa oraz całego tekstu. W czasie działania programu, w pamięci przechowywany jest currentWord, czyli aktualne słowo tj. ciąg znaków, liter lub cyfr, które zaczynają się na początku wpisywanego tekstu lub po spacji, a kończą się w pozycji kursora. Dodanie spacji po ciągu znaków kończy słowo i usuwa je ze zmiennej currentWord, a dodaje do zmiennej zwanej wholeText, która przechowuje dotychczas wpisany tekst. Przykładowo jeśli mamy tekst jak na rysunku 1.5 to w zmiennej wholeText przechowywujemy ””Wymagajcie od siebie choćby inni od was nie wymagali.” Jan Paweł II ”, a w currentWord ”sie”. W ten sposób podpowiedzi generowane będą jedynie dla cząstki ”sie”, a tekst wpisywany będzie w pozycji kursora, która również monitorowana jest przez zmienną currentPosition. Scalenie wholeText i currenWord następuję, gdy zmieniamy pozycję kursora strzałkami, lub jeśli do tekstu dodana zostanie spacja, a za kursorem nie znajduje się żaden znak. Przykład

przedstawia rysunek 1.6.



Rys. 1.5 – Przykładowe zapisywanie tekstu do zmiennych w zależności od pozycji kursora.



Rys. 1.6 – Przykładowe zapisywanie tekstu do zmiennych w zależności od pozycji kursora.

Zmienna currentWord została zespolona z wholeText poprzez wklenie jej na pozycji zapisanej jako currentWordStart. W celu dynamicznego ustalania pozycji currentWordStart oraz zawartości currentWord powstały funkcje: getCurrentWordStart() oraz getCurrentWord(). Działanie pierwszej zademonstrowano za pomocą pseudokodu 2.

Algorithm 2 Działanie funkcji getCurrentWordStart()

```
if currentWord nie jest pusty && currentPosition nie jest na początku tekstu
then
  for każda pozycja aż do początku tekstu do
    pobierz literę z wholeText w danej pozycji
    if pobrana wartość nie jest liczbą ani literą then
      currentWordStart = danapozycja + 1
    end if
  end for
end if
```

Działanie drugiej sprowadza się do wycięcia faragmentu tekstu między currentWordStart, zaimplementowanym w wyniku działania poprzedniej funkcji, a currentPosition.

1.3.2.1 Usuwanie znaków

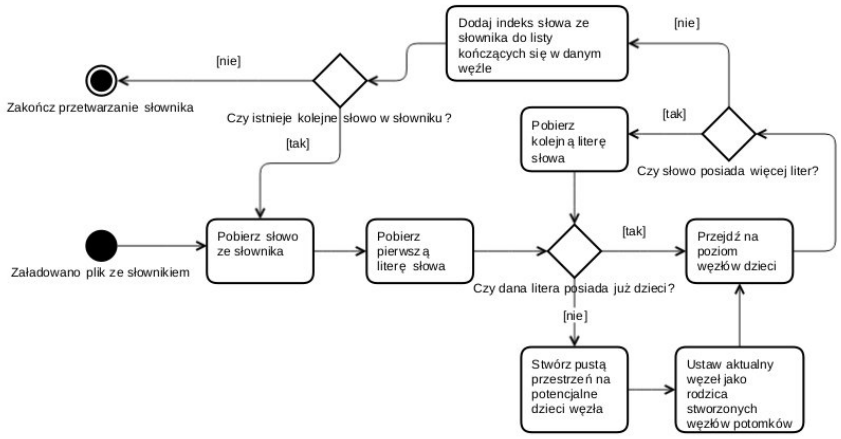
Proces usuwania znaków staje się trudniejszy ze względu na ułatwiające kontrolę nad tekstem currentWord i wholeText. Pierwszym napotkanym problemem była sytuacja, w której currentWord jest puste. W takim wypadku należy usunąć znak znajdujący się w currentPosition ze zmiennej wholeText, a następnie za pomocą wcześniej opisanych funkcji getCurrentWordStart() oraz getCurrentWord() otrzymać informację o nowej wartości currentWord. Kolejnym krokiem jest wycięcie wartości zmiennej currentWord z wholeText, przesunięcie kursora dla informacji użytkownika w odpowiednią pozycję oraz odświeżenie wyglądu

podpowiedzi. Gdy currentWord ma wartość, sytuacja upraszcza się do usunięcia ostatniego znaku z currentWord. W celu reprezentacji tekstu dla użytkownika należy scalić wholeText z currentWord, ale by nie utracić wartości wholeText stworzono tymczasową zmienną currentText, który zawiera całą treść wpisanego tekstu.

1.3.3 Trie tree

Jak wyżej **??** wspomniano, w pracy wykorzystano drzewo typu Trie w celu pracy z rozległym słownikiem. Słowa z pliku w formacie TXT wczytywane są do programu podczas uruchamiania klawiatury- każda z linii dostarczanego pliku powinna stanowić pojedynczy wyraz. Słowa te, za pomocą sztucznie wygenerowanych list kodujących oraz dekodujących polski alfabet, wprowadzane są do drzewa typu Trie. Drzewo takie powstaje poprzez stworzenie pustego węzła typu node - struktury zadeklarowanej w pliku nagłówkowym klasy obsługującej współpracę ze słownikiem – Dictionary. Struktura node przechowuje informacje o rodzicu bieżącego węzła, o jego potomkach – czyli węzłach następujących oraz wektor zawierający informację o przynależności danego węzła literowego do słowa. Po zainicjowaniu węzła zerowego, po kolei analizowane są słowa ze słownika w funkcji insertWord().Każde jest rozpatrywane jako tablica liter (char) i iteracyjnie następuje najpierw kodowanie litery na odpowiadający jej indeks (za pomocą mapy alfabet), potem, sprawdzane jest, czy w drzewie nie istnieje już węzeł odpowiadający danej wartości. Gdy nie znaleziono gałęzi drzewa pasujących do poszukiwanej wartości tworzy się za pomocą funkcji calloc przestrzeń w pamięci na przyszłe dzieci tego węzła, a jako ich rodzica podaje się aktualnie przeglądany węzeł z literą. Kolejno, niezależnie od wyniku uprzednio rozpatrywanego warunku, o ile słowo wciąż posiada litery do przeglądu, przenosi się o poziom niżej w drzewie (na poziom dziecka uprzednio rozpatrywanej litery) i proces zachodzi od początku. Gdy przetworzono wszystkie litery słowa do listy wystąpień ostatniego odwiedzonego węzła dopisany zostaje indeks słowa w słowniku – tym samym oznaczając jego koniec. Na rysunku 1.7 przedstawiono, w sposób graficzny, działanie danej funkcji.

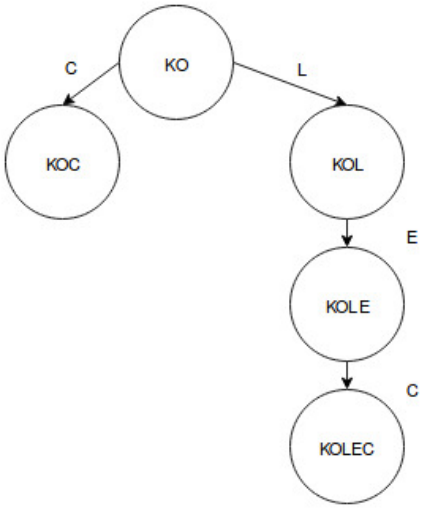
Po uzupełnieniu słownika nastąpić może autouzupełnianie wpisywanego tekstu. Każde wpisanie litery powoduje wywołanie metody updateHints(), która jest odpowiedzialna za tworzenie oraz wyświetlanie podpowiedzi, zgodnie z wprowadzonym do tej pory słowem. Jako poszukiwaną frazę traktujemy ciąg znaków, które użytkownik wpisał do pozycji kursora od ostatniej spacji, bądź początku tekstu. W wypadku, gdy ten ciąg znaków jest dłuższy niż dwa, wykorzystywana jest funkcja komunikująca się ze stworzonym słownikiem – searchWord(). Przekazywane jest drzewo Trie słownika oraz aktualnie poszukiwana fraza (bez formatowania). Wpisywana fraza, również traktowana jest jako zbiór liter, które przeglądane są jedna po drugiej. Dla każdej następuje zmiana dzięki mapie kodującej alfabet na odpowiadający indeks, który umożliwia przeszukiwanie słownika. Sprawdzane jest, czy istnieje potomek drzewa Trie o danym indeksie – jeśli tak, to następuje zmiana węzła na węzeł dziecka, tak, że przy przeszukiwaniu drzewa będą brane pod uwagę jedynie węzły z rodzicem będącym pierwszą literą poszukiwanej frazy. Gdy przeszuka się wszystkie litery, bądź w trakcie tego procesu zabraknie węzłów potomków dla danej kombinacji liter to zwracany jest odpowiednio ostatni węzeł wspólny dla danej frazy lub też



Rys. 1.7 – Diagram przepływu dla funkcji insertWord() wprowadzającej słowa do drzewa typu Trie.

węzeł pusty.

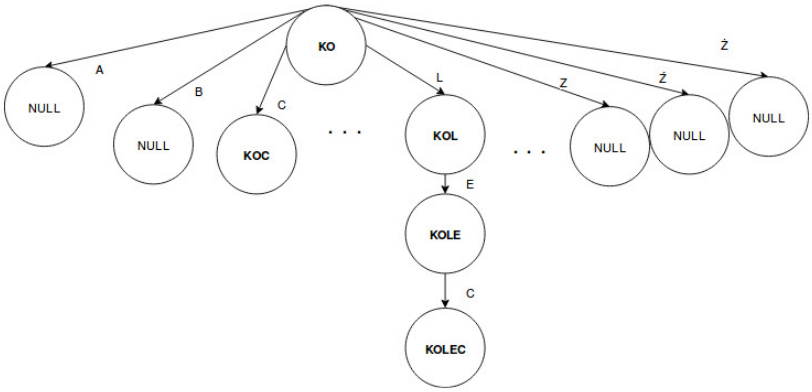
W celu lepszego zrozumienia działania algorytmu rozpatrzmy go na przykładzie. Załóżmy, że mamy drzewo takie jak na rysunku **??**. Jeśli wyszukamy frazę ”ko” to funkcja zwróci nam węzeł i jego dzieci. Możliwe autouzupełnienia wyglądają wtedy tak jak na rysunku 1.8. Złożenie końcówek wyrazów zachodzi w



Rys. 1.8 – Przykładowy węzeł do autouzupełniania słów po wpisaniu frazy ”ko”.

funkcji getSimilarEndings(), której przekazywany jest znaleziony węzeł, pusty wektor, do którego mają zostać zapisane wynikowe wyrazy oraz węzeł znaków niezbędny do dekodowania. W pierwszym kroku sprawdzane jest, czy w danym węźle kończą się jakieś słowa, jeśli tak (wektor wystąpień węzła jest różny od zera),

to każdą literę zapisaną w wyżej wymienionym wektorze znaków zapisuje się do jednego słowa (tworząc końcówkę do autouzupełniania). Gotowy ciąg znaków zapisywany jest do wektora z końcówkami. W wypadku pierwszego wykonania się funkcji mamy do czynienia z pustym wektorem znaków- toteż nie zostanie stworzona żadna końcówka. Nawet jeśli ''ko" było pełną formą słowa nie powinna się ona wyświetlać w proponowanych opcjach użytkownika. Aby uzupełnić wektor znaków należy przejrzeć przesłany węzeł (ten z rysunku 1.8) – w tym celu sprawdza się, czy dzieci węzła odpowiadającej każdej literze alfabetu nie są puste, gdyż programowe drzewo ma, prócz wcześniej przedstawionych gałęzi, jeszcze 33 (zakładając, że zaimplementowany alfabet posiada 35 liter) nieobdarzone wartością gałęzie przedstawione na rysunku 1.9. Gdy znaleziono element o niezerowej wartości pobierana jest za pomocą mapy symetrycznej (reverseAlfabet) wartość literowa węzła i wprowadzana jest do wektora znaków. Węzeł z ''ko" zmieniany jest w węzeł pierwszego pierwszego dziecka – w tym wypadku ''koc". Następnie przez rekurencję ponownie rozpoczyna się sprawdzanie, czy dane słowo kończy się w tym węźle. Jeśli tak, to proces zachodzi według powyżej opisanych kroków, jeśli nie, to znów poszukiwany jest węzeł potomny z kolejną literą końcówki słowa do autouzupełniania. Algorytm przedstawiono w sposób graficzny na rysunku 1.10

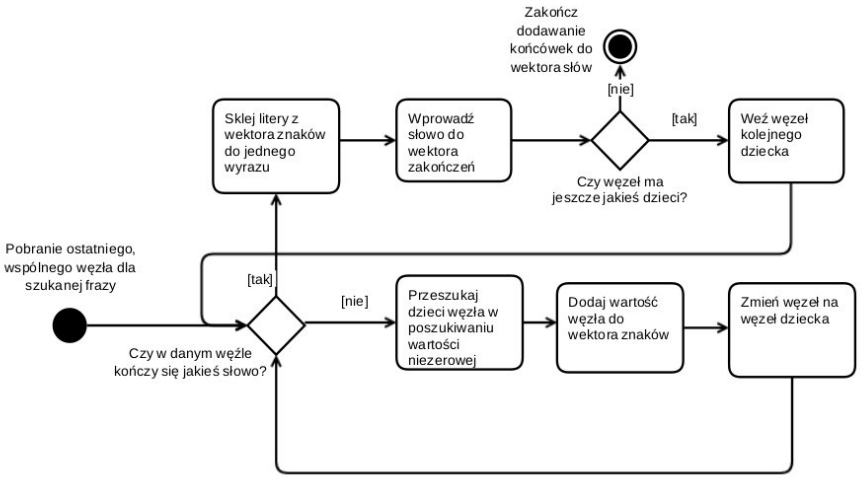


Rys. 1.9 – Reprezentacja przykładowego węzła widzianego w programie.

Ostatnim krokiem w stworzeniu podpowiedzi jest skrócenie listy końcówek do liczby przycisków przeznaczonych na podpowiedzi oraz zespolenie ich z dotychczas wpisanym słowem. Taki ciąg znaków można przedstawić użytkownikowi jako napis na przycisku, który po wybraniu wpisuje reprezentowany tekst do pola tekstowego zamieniając dotychczasową frazę na wybraną oraz dodając znak spacji na końcu nowowybranego słowa.

1.3.4 Dynamiczna zmiana czasu progowego fiksacji

Jak wspomniano wcześniej klawiatura umożliwia dynamiczną zmianę czasu czasu progowego fiksacji (decydującego o tym kiedy dany przycisk zostanie wywołany). Zmiana następuję w oparciu o ilość błędów wykonanych przez użytkownika w określonym oknie czasowym. Weryfikacja następuję co minutę. Jeśli w tym czasie użytkownik popełni 5 lub więcej błedów (użyje przycisku Backspace), to czas



Rys. 1.10 – Diagram przepływu dla funkcji getSmiliarEndings().

fiksacji ulegnie wydłużeniu o jedną sekundę. Pięć błedów stanowi, dla ustawień początkowych, czyli progowego czasu fiksacji ustawionego na 3s, 25% znaków wpisanych w tym czasie. Dla ilości błędów mniejszej niż dwa czas fiksacji zostaje skrócony. Jeśli użtkownik użyje przycisku Backspace 3-4 razy w ciągu minut czas progowy fiksacji nie ulegnie zmianie. Czas fiksacji został ograniczony obustronnie poprzez stałe z pliku const.h. Założono, że nie może on być krótszy niż 1s i dłuższy niż 6s. Użytkownik ma również możliwość manualnego ustawienia czasu progowego, poprzez zmianę wartości w oknie menu.

1.3.5 Korzystanie z wyszukiwarki internetowej

W celu pracy z przeglądarką niezbędne jest podłączenie drugiego ekranu, na którym może się otwierać okno przeglądarki. W innym wypadku okno klawiatury ulegnie minimalizacji i nie ma możliwości powrotu do okna.

W przypadku skorzystania z jednego z trybów wysłania (”Google”, ”YouTube”, ”Filmweb”) wywoływana jest metoda, GoogleSearcher, search(). W zależności od przesłanej wartości sendingState, informującego o tym, gdzie wysłane ma być zapytanie, do wyszukiwanego tekstu (pobranego z pola tekstowego) dołączana jest informacja, w którym serwisie szukać wyników. Taka informacja załączana jest jako argument uzupełniający URL powstały przez stworzenie spersonalizownej wyszukiwarki, dzięki specjalnemu Google API. Następnie obiekt klasy QNetworkAccessManager wywołuję metodę RESTową GET na obiekcie klasy QNetworkRequest, który za argument konstruktora przyjmuje powstałe URL z paramatrem.

1.3.5.1 Google Api

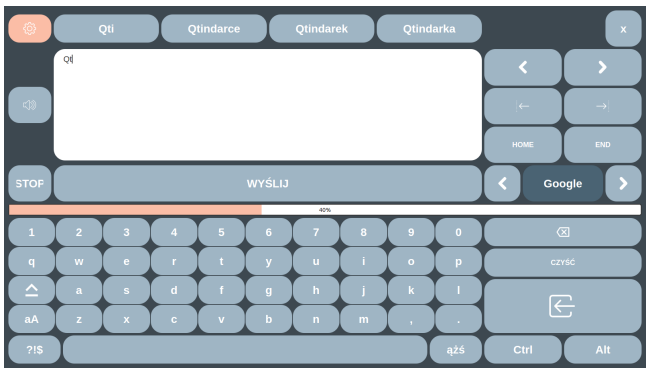
Jak już wcześniej wspomniano **??** do projektu wykorzystano specjalne API Google - Google Custom Search Engine. Dzięki temu powstał specjalny link

URL umożliwiający przyjmowanie różnych parametrów jako zapytanie do wyszukiwaraki. Co więcej specjalne API na rządanie GET zwraca iformację w postaci QNetworkReply, gdzie funkcja handleNetworkData() umożliwia jego przetworzenie w ustrukturyzowaną postać JSON. Obiekt JSON zawiera 10 pierwszych wynków wyszukiwania w specjalnej przeglądarce. Każdy z obiektów typu JSON posiada informacje o wyniku wyszukiwania. W skład takiego obiektu wchodzą **?**:

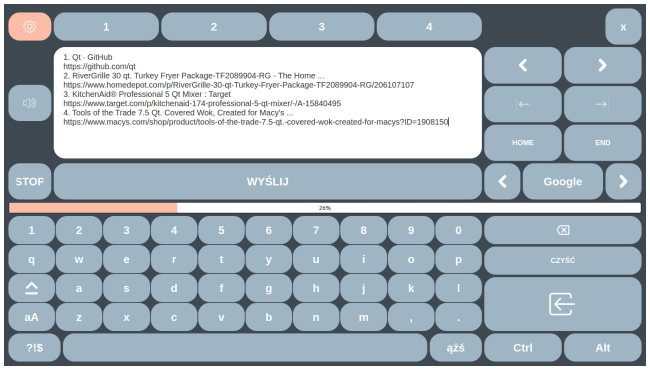
- ”kind”: ”customsearch#result”
- ”title”: string
- ”htmlTitle”: string
- ”link”: string
- ”displayLink”: string
- ”snippet”: string
- ”htmlSnippet”: string
- ”cacheId”: string
- ”mime”: string,
- ”fileFormat”: string
- ”formattedUrl”: string
- ”htmlFormattedUrl”: string
- ”pagemap”

Wykorzystując dane z ”title” oraz ”link” przedstawiane są użytkownikowi cztery pierwsze wyniki wyszukiwania w polu tekstowym,a ich wywołanie (otworzenie strony) odbywa się przez wybranie przystosowanych przycisków podpowiedzi z numerem wyniku wyszukiwania w przypadku wyszukiwania danych w Google. Przykład działania przedstawiono na rysunkach 1.11 oraz 1.12.

W wypadku wyszukiwań w Youtube oraz Filmweb użytkownik nie ma możliwości wyboru wyniku wyszukiwania - otwierany jest pierwszy wynik, który zawiera odpowiednie słowa kluczowe.



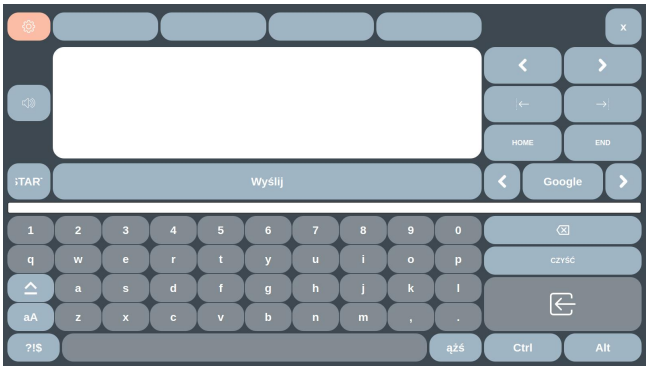
Rys. 1.11 – Przykładowy tekst wpisany do pola tekstowego, wysyłany jako argumnet wyszukiwania Google.



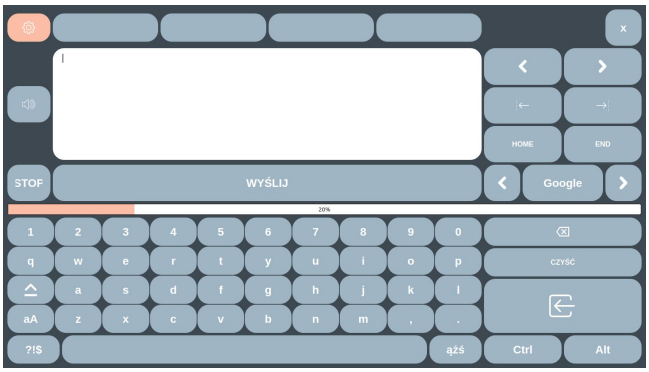
Rys. 1.12 – Wyniki wyszukiwana tekstu wpisanego na rysunku 1.11.

1.4 Zaimplementowany interfejs użytkownika

Zaimplementowany interfejs użytkownika uległ przemodelowaniu w porównaniu z przedstawionym wcześniej projektem. **??** Wygląd nowego interfejsu można zaobserwować na rysunkach 1.13, 1.14, 1.15. Analizując widok klawiatury od górnego lewego rogu, pierwszą zmianą, którą da się zauważyć to zmina napisu ”menu” na intuicyjną ikonę, która w sposób bardziej spójny pasuje do całego wyglądu. Wszystkie ikony projektu pobrane zostały ze strony **?** i udostępniane są na podstawie licencji CC 3.0. Autorami są Smashicons, D. Grandy, C. Fertu oraz G. Cresnar. Kolejną zmianą jest rozmieszczenie przycisków *home*, *end* oraz *czyść*. Zostały one posortowane tematycznie, tak by użytkownikownik mógł je łatwiej odszukać. W ich miejsce przestawiony został przyciski Text-ToSpeech (aktualnie nieobsługiwany). W miejsce strzałek służących do poruszania się w kierunach góra-dół tekstu stworzono przyciski, których zadaniem jest przemieszczanie się po tekście o jedno słowo. Podczas testów zauważono, że przy pisaniu niedługich fragmentach tekstu, do których prawdopodnie najczęściej wykorzystywana będzie klawiatura, są one bardziej wykorzystywane niż strzałki góra-dół. Dużym ułatwieniem dla użytkownika była również zmiana dotychczas niewielkiego, kolistego paska postępu na poprzeczny pasek, znajdu-



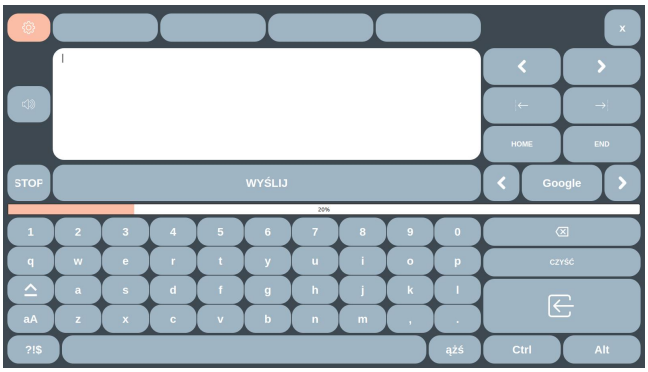
Rys. 1.13 – Widok interfejsu, gdy przyciski są w trybie wstrzymania.



Rys. 1.14 – Widok interfejsu, gdy przyciski są w trybie aktywnym.

Rys. 1.15 – Widok okna menu.

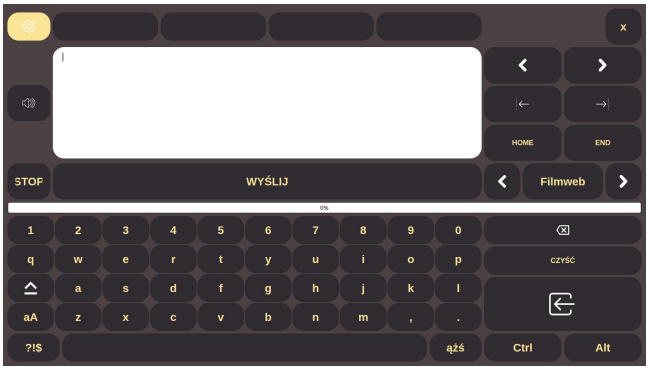
Rys. 1.16 – Możliwe widoki trybów wyszukiwania.



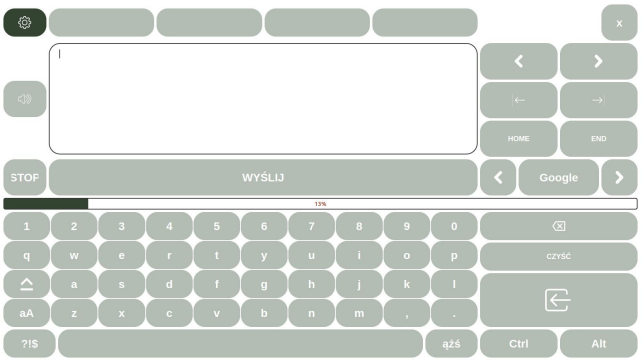
Rys. 1.17 – Widok interfejsu w trybie *Basic*.

jący się pod przyciskiem wyślij. Łatwiej w ten sposób obserwować zmieniajacy się jego stan bez odrywania wzroku od fiksowanego przycisku. Dodatkowo pasek jest w kolorze kontrastującym z barwą innych przycisków. Przycisk *wyślij* celowo został zaprojektowany jako największy w klawiaturze, by był łatwy do trafiena i dostęp do niego był wygodny podczas konwersacji przez broadcast. Obok niego pojawiły się przyciski zmieniające tryby wyszukiwania oraz przycisk dostosowujący swój napis do wybranego trybu. Zmiana wygląda tak jak na rysunku 1.16.

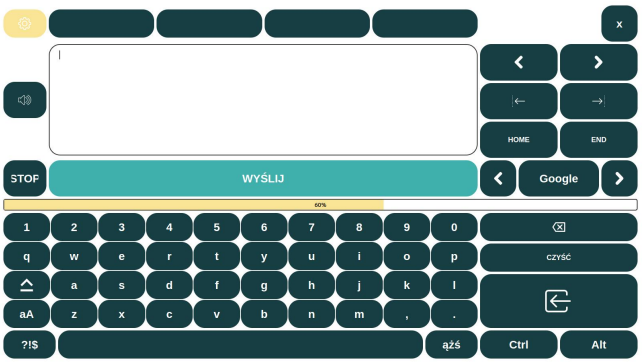
Jak widać na rysunku 1.15 użytkownikowi do wyboru przedstawiono 5 wersji kolorystycznych interfejsu 1.17, 1.18, 1.19, 1.20, 1.21. Kolory dobrano tak, by stanowiły przyjemną do oka, spójną wizję aplikacji, a jednocześnie dawały wystaczający kontrast do wydajnej pracy. Paletę *Dark* zaprojektowano z myślą o pracy przy słabym oświetleniu, tak by jasne kolory ekranu nie raziły użytkownika w oczy. Paleta *Light* przystosowana jest dla użytkowników o potrzebie mniejszego kontrastu. Zarówno kolor zielony jak i niebieski są uważane według autorów publikacji **?** za barwy o charakterze relaksującym. Palety *Modern* oraz *Wine* powstały w celu zwiększenia możliwości wyboru, gdyż ostateczna ocena wyglądu intefejsu zależy od subiektywnych odczuć użytkownika. Wszystkie wyżej wymienione palety powstawły w oparciu o palety stworzone na stronie **?**.



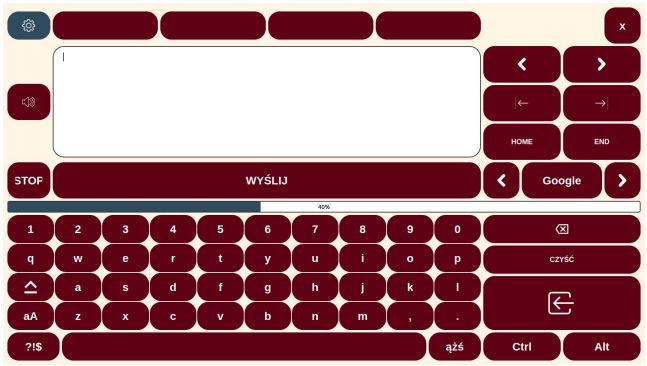
Rys. 1.18 – Widok interfejsu w trybie *Dark*.



Rys. 1.19 – Widok interfejsu w trybie *Light*.



Rys. 1.20 – Widok interfejsu w trybie *Modern*.



Rys. 1.21 – Widok interfejsu w trybie *Wine*.

1.5 Podsumowanie

W ramach relizacji projektu udało się zaimplementować wszystkie zaplanowane w rodziale **??** funkcjonalności. Niektóre zachowania zostały zmienione (np. wybór trybu wysyłania/wyszukiwania) względem zaplanowanych na diagramie przypadków użycia **??**. Wynikło to z analizy sposobu korzystania z przycisków klawiatury i uznano nowy rozkład za bardziej ergonomiczny. Działanie oprogramowania omówiono w powyższym rozdziale ze szczegółowym opisem zastosowanych algorytmów bazowych, dzięki którym możliwe jest poprawne działanie aplikacji. Wymieniono i wytłumaczono rolę najważniejszych zmiennych. Omówiono nowy wygląd intefrejsu użytkownika oraz powody, dla których wprowadzono różnice w porównaniu z prototypem.