Architektura Systemów Komputerowych – labolatorium nr 2

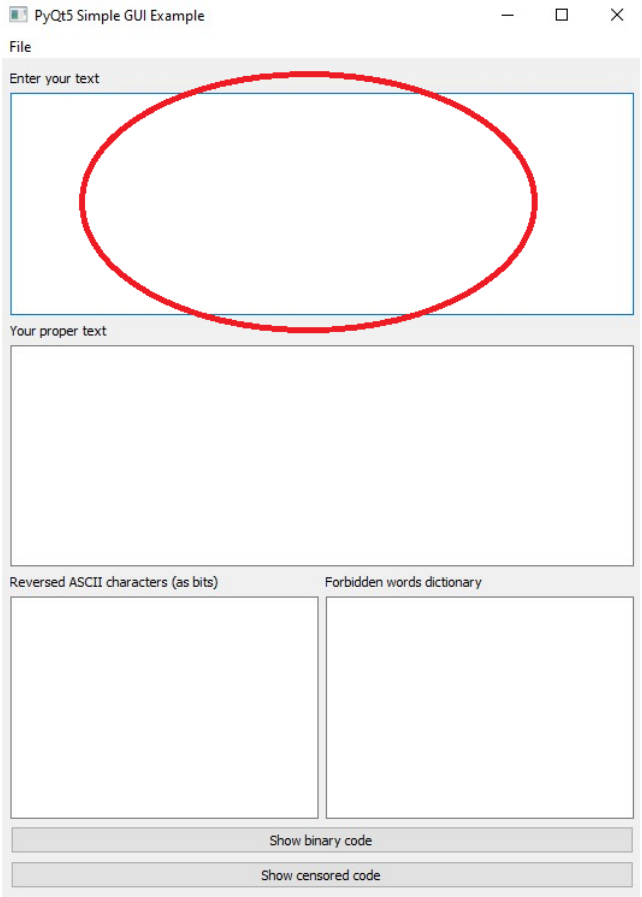
Tymoteusz Byrwa 184414

Jakub Kłopotek-Główczewski 186067

Zadanie labolatoryjne polegało na stworzeniu aplikacji symulującej przesył danych zgodny ze standardem rs232. Nadajnik tekstu oraz odbiornik są umieszczone w jednym oknie aplikacji, a nośnikiem danych jest tablica.

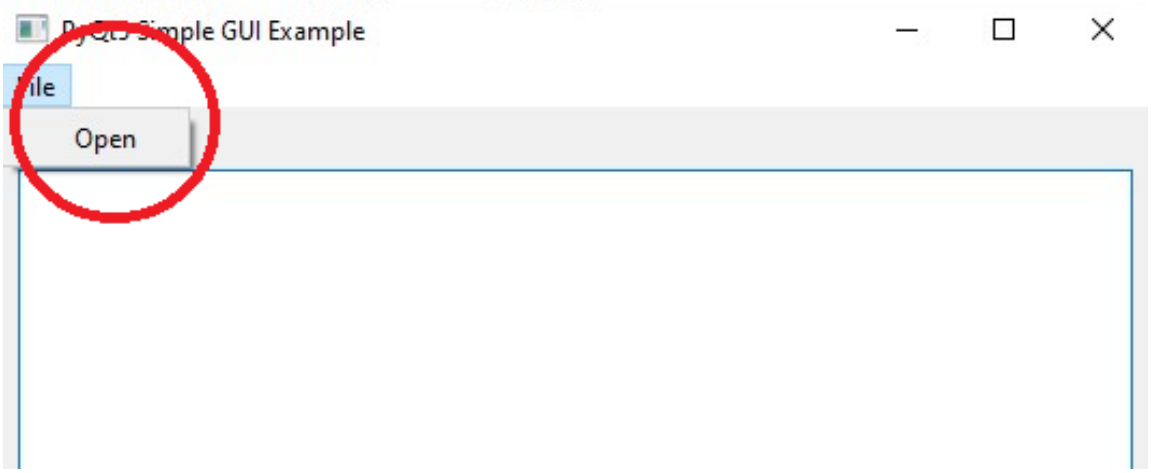
Plik .exe znajduje się w katalogu /dist.

Powstała aplikacja pobiera od użytkownika tekst w oknie:



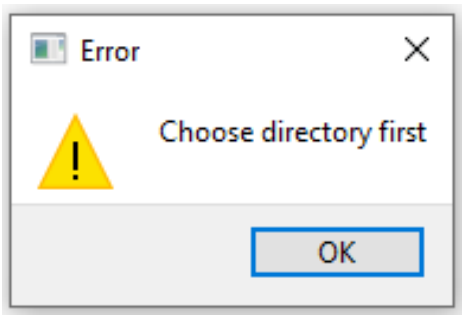
Tekst widziany jest oczami programu jako “*plain text*”.

Następnie użytkownik powinien wybrać plik ze słownikiem zawierającym słowa podlegające cenzurze:



Po wybraniu tej opcji pojawi się okno systemu Windows do wyboru plików txt.

W przypadku gdy użytkownik spróbuje ominąć wybór słownika i będzie chciał otrzymać ocenzurowany tekst od razu, akcja ta zostanie przejęta przez obsługę wyjątków w efekcie czego wyświetlony zostanie następujący komunikat:

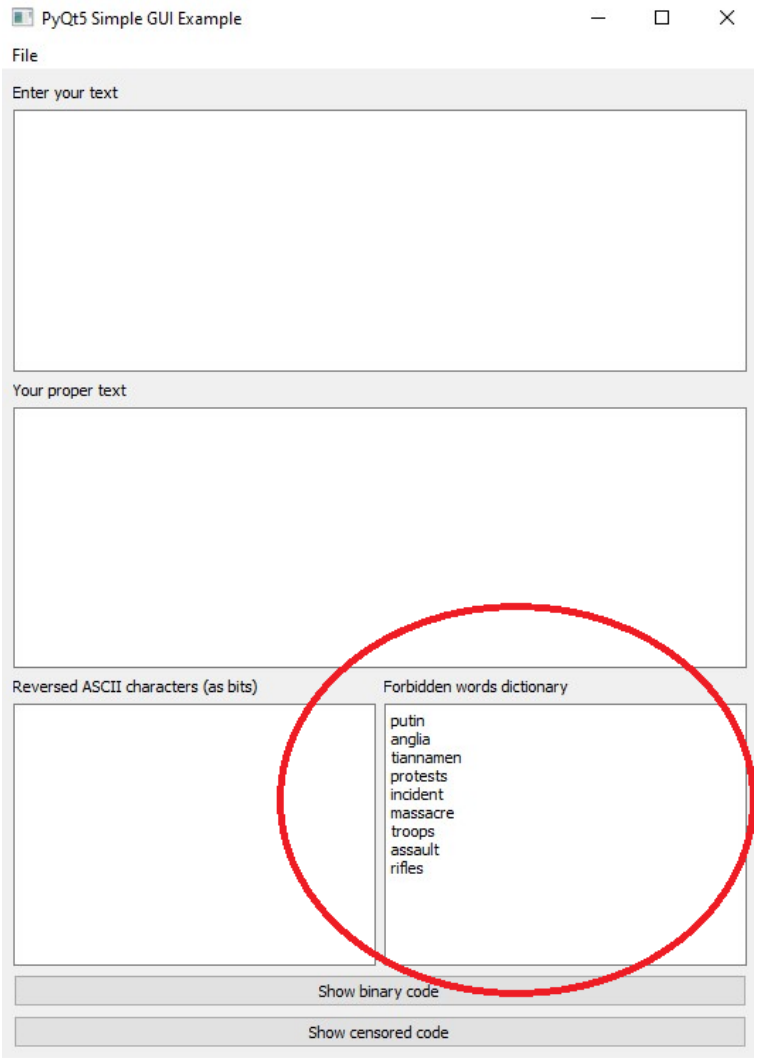


Tak wygląda obsługa wyjątku w kodzie programu:

```
def update_box2(self):
    try:
        text = self.text_input.toPlainText()
        censored_text = self.censor_text(text.lower())
        self.display_box2.setPlainText(censored_text)
    except AttributeError:  # Handle empty dictionary
        QMessageBox.warning(self, "Error", "Choose directory first")
        return
```

W przypadku gdy nie ma instancji listy zakazanych słów, metoda ***self.censor_text()*** nie jest w stanie dokonać cenzury, co skutkuje błędem ***AttributeError*** i wyświetleniem ostrzeżenia.

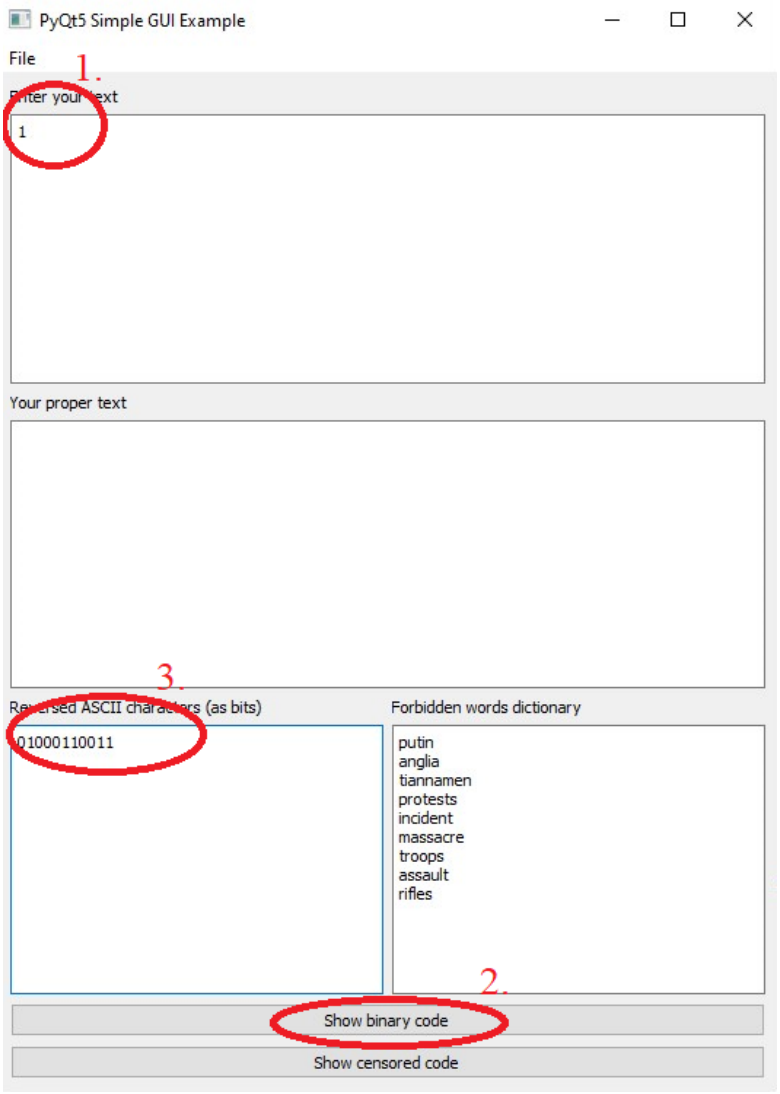
Gdy użytkownik wybierze interesujący go słownik, w oknie pojawiają się słowa podlegające cenzurze:



Jeśli użytkownik poda interesującą go frazę, a następnie kliknie przycisk ***"Show binary code"***, w oknie pojawi się sekwencja zer i jedynek (bitów), reprezentujących odwrócony ciąg znaków ASCII. Przykładowo z następującymi, arbitralnie przyjętymi flagami:

- LSB: 0
- MSB: 11,

dla znaku 1 (*w zapisie 8-bitowym: 00110001*) mamy odwrócony ciąg binarny z dodanymi flagami:



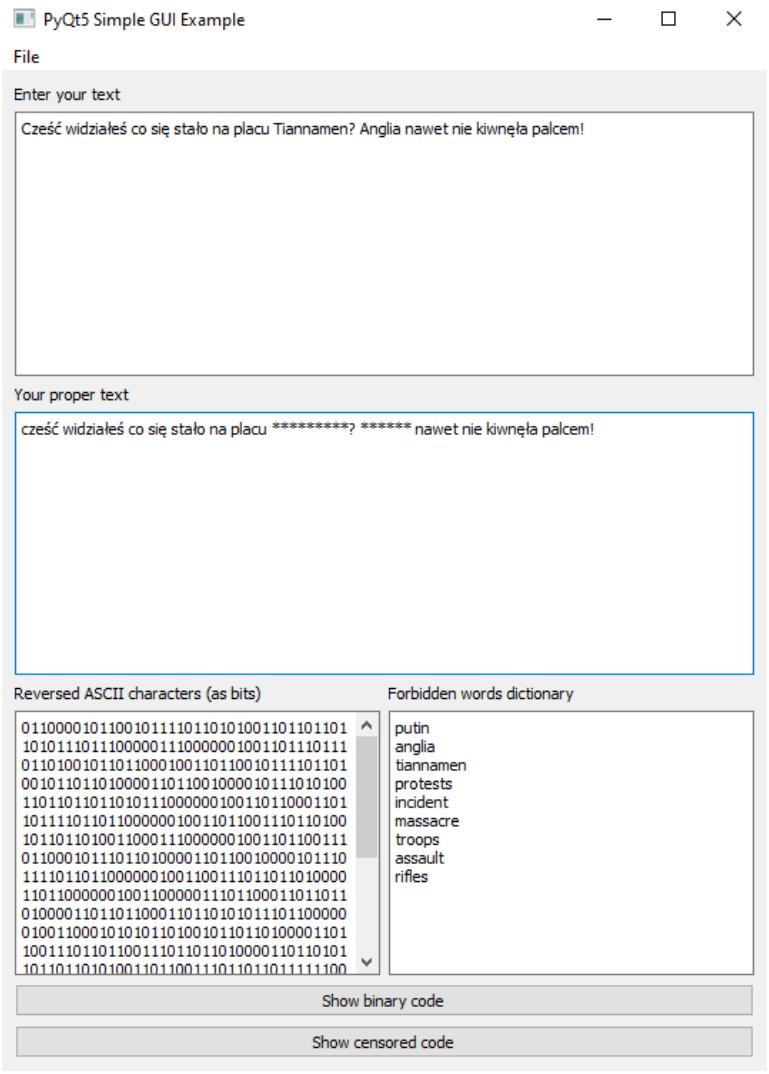
W kodzie programu enkodowanie jest realizowane za pomocą następującej funkcji:

```
def text_to_serial_string(text):
    start_bit = '0'
    stop_bits = '11'
    serial_string = ""

    for char in text:
        # Convert to 8-bit binary and reverse the order (LSB to MSB)
        char_bits = format(ord(char), '08b')[::-1]
        serial_char = start_bit + char_bits + stop_bits
        serial_string += serial_char

    return serial_string
```

Bity startu i stopu są tutaj narzucone przez programistę, a kodowanie znaków odbywa się za pomocą funkcji złożonej: ***format(ord(char), '08b')[::-1]***.

Funkcja ***ord(char)*** pobiera znak i zwraca jego Unicode, a funkcja ***format(_, '08b')*** dodatkowo zamienia Unicode na ciąg 8-bitowy, dzięki argumentowi *'08b'*.

Gdy finalnie użytkownik chce zobaczyć wersję swojego tekstu pozbawioną nieodpowiednich słów, może kliknąć przycisk ***“Show censored code”***:



Co warte uwagi, słownik docelowo zawiera słowa pisane małymi literami (lower case), funkcja pobierająca tekst od użytkownika przekształca go w programie na taki sam format, przez co wielkość liter nie ma znaczenia.

Cenzurowanie słów znajdujących się w słowniku odbywa się na etapie enkodowania:

```
def censor_text(self, input_text):
    encoded_input_text = text_to_serial_string(input_text)
    words_to_censor = serial_string_to_text(
        self.encoded_file_content).split()

    for word in words_to_censor:
        encoded_word = text_to_serial_string(word)
        if encoded_word in encoded_input_text:
            censor_string = '*' * len(word)
            input_text = input_text.replace(word, censor_string)

    return input_text
```

Jeśli odpowiedni ciąg bitów zostanie odnaleziony w sekwencji, słowo to jest zamieniane w nośniku danych (tablicy słów), ciągiem znaków ‘*’ o długości danego słowa.

Dekodowanie słów w nośniku jest realizowane w następujący sposób:

```
def serial_string_to_text(serial_string):
    start_bit = '0'
    stop_bits = '11'

    text = ""
    index = 0
    while index < len(serial_string):
        if serial_string[index] == start_bit:  # Check for start bit
            # Extract character bits (still reversed)
            char_bits_reversed = serial_string[index + 1:index + 9]
            # Reverse the order back to the original (MSB to LSB)
            char_bits = char_bits_reversed[::-1]
            char = chr(int(char_bits, 2))  # Convert binary to ASCII character
            text += char
            # Move to the next character (1 start bit + 8 character bits + 2 stop bits)
            index += 11
        else:
            raise ValueError("Invalid bit stream format")

    return text
```

- Znalezienie bitu startowego
- Wyodrębnienie słowa o długości 8 bitów
- Odwrócenie ciągu bitów
- Zamiana bitów na ASCII
- Dodanie ciągu znaków ASCII to sentencji
- Przesunięcie wskaźnika na listę bitów o zadaną wartość

Dyskusja

Napisany program działa szybko i realizuje założenia ćwiczenia labolatoryjnego. Aplikacja daje duże możliwości edycji cenzury. Poprzez składowanie kluczowych słów w plikach tekstowych, użytkownik może wykluczyć dużą liczbę słów. Ponadto możliwość wyboru różnych plików tekstowych umożliwia użytkownikowi ich podział tematyczny, w zależności od potrzeb (jeden słownik może na przykład zawierać słowa związane z polityką, a inny związane ze sportem).

Program można w łatwy sposób rozwinąć, tak żeby automatycznie cenzurował dokumenty tekstowe i automatycznie generował poprawne wersje (na przykład w formacie pdf), co wymaga jednak poza zakres tego ćwiczenia.