Raport projektu z KCK „Speller”

Franciszek Połencki, Gracjan Popiółkowski

Wszystkie pliki wspominane w raporcie, są zapakowane w archiwum „Projekt_pliki.zip”.

1. Opis zadania:

Zadaniem projektu jest stworzenie prostego interfejsu offline typu "speller" opartego na mruganiu. Projekt realizuje się w grupach 2-osobowych.

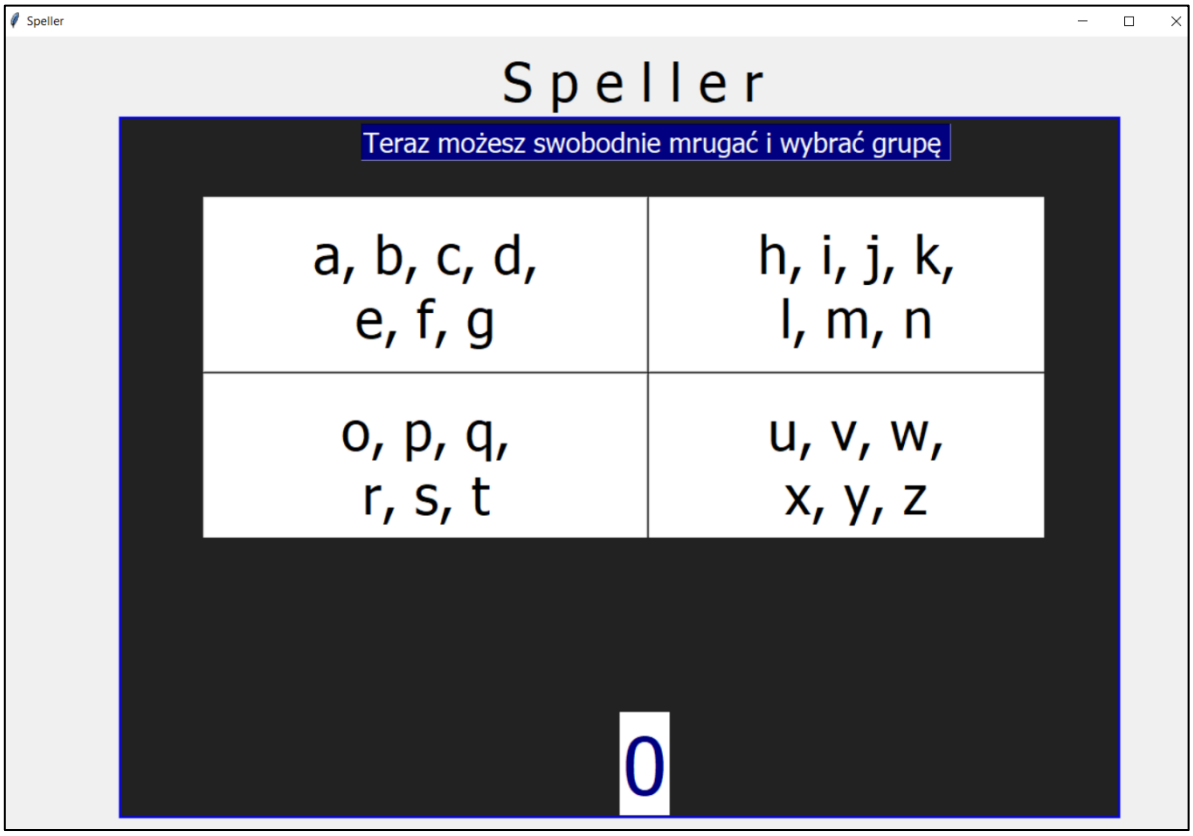
W ramach projektu wymagane jest:

A. przygotowanie programu wyświetlającego litery alfabetu,
B. zebranie danych osoby, która przy pomocy mrugania sygnalizuje wybór danej litery (powinna zapisać w ten sposób jakiegoś słowo),
C. przygotowanie kodu, który odszyfrowuje wybrane litery, oraz
D. przygotowanie raportu z projektu.

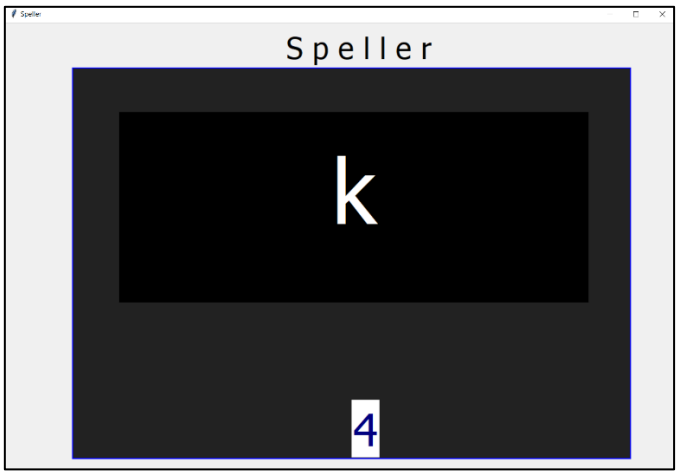
2. Wybrane metody:

Program wyświetlający litery:

Stworzyliśmy program graficzny (Speller.py), przy użyciu biblioteki tkinter, który polegał na wyborze łańcucha wyświetlanych liter, spośród czterech sześcio lub siedmioliterowych grup, za pomocą kliknięcia kursorem myszy na odpowiedni przycisk.



Po wybraniu odpowiedniej grupy, litery w kolejności alfabetycznej, pojawiały się na ekranie na okres 1000ms. Osoba, która zapisywała za ich pomocą słowa, miała za zadanie wybierać odpowiednią grupę i mrugnąć, kiedy na ekranie pojawiła się następna litera z kodowanego słowa. Następne mrugania podczas przelatywania ciągu nie były już liczone, ponieważ ustaliliśmy, że zawsze jedno wybranie grupy odpowiada jednej zakodowanej literze, nawet jak następna litera słowa akurat znajdywała by się w dalszej części ciągu.



Cyfra w dolnej części programu jest oznaczeniem, która litera ciągu aktualnie wyświetla się na ekranie, nie znalazło to jednak żadnego zastosowania. A po wyświetleniu całego ciągu program wracał do ekranu startowego (panel z wyborem grup), gdzie umieściliśmy informację „Teraz możesz swobodnie mrugać i wybrać grupę”.

Sposób zapisywania danych:

Program dodatkowo zapisywał nam dane w konsoli. Po każdym kliknięciu na grupę zapisywał, w czterech linijkach:

1. Grupę, jaka została wybrana,

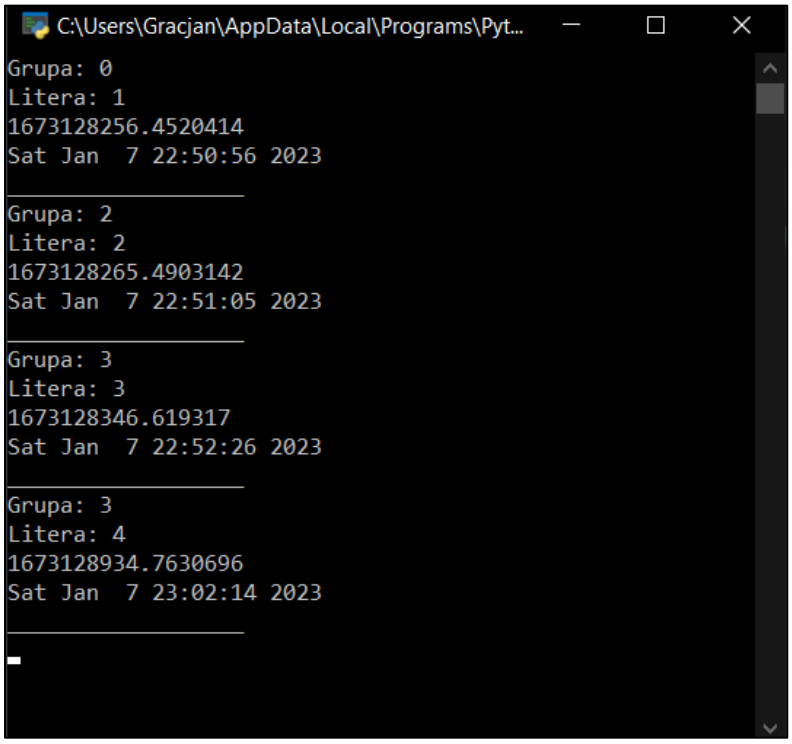
2. Która to jest litera kodowanego słowa (następne słowo było, kodowane po zresetowaniu programu),

3. Czas jako liczbę sekund, które upłynęły od 1 stycznia 1970 roku 00:00:00 czasu uniwersalnego,

4. Czas z datą i godziną.

Po każdum słowie dane były ręcznie przekopiowywane do zewnętrznego pliku „DaneZKCK.txt”.

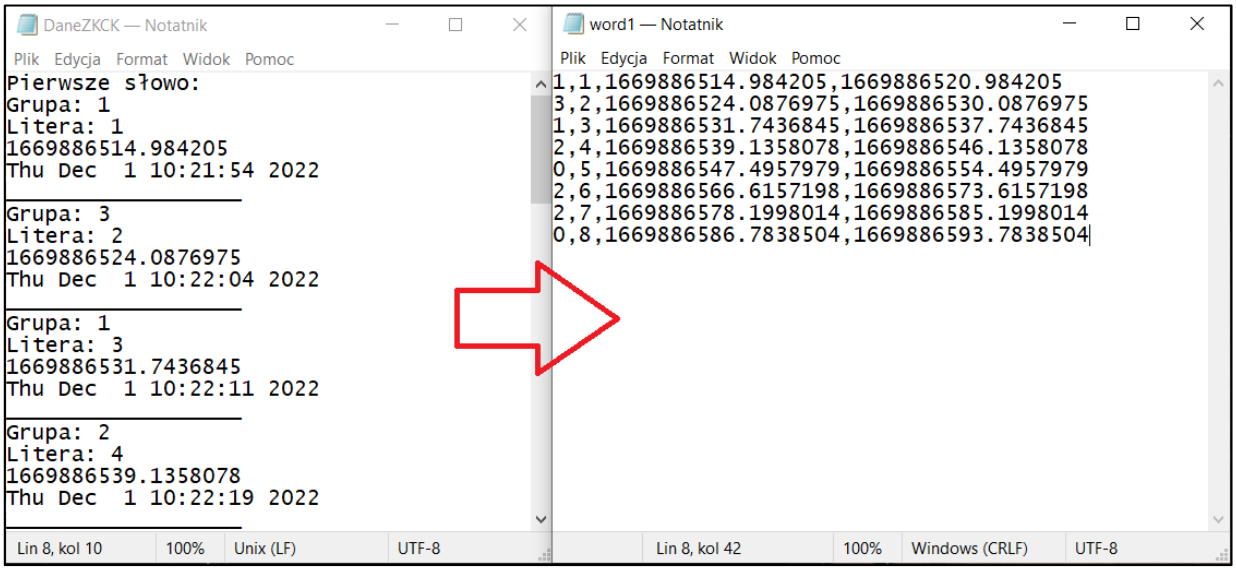
W dzień zbierania danych do projektu wymrugaliśmy dwa słowa „owsianka” i „Japonia”.



3. Opis analizy i uzyskanych wyników:

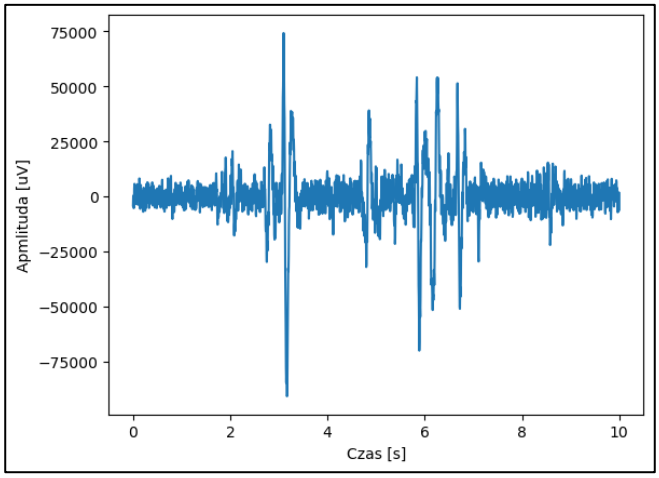
Cały program i kroki pomocnicze do analizy sygnału są zapisane w pliku „Analiza.ipynb”.

Nasz plik z danymi „DaneZKCK.txt” przerobiliśmy na format csv korzystając z excela i utworzyliśmy z tego dwa oddzielne pliki na nasze dwa zakodowane słowa „word1.txt” i „word2.txt”, by następnie wczytać te dane do naszego programu w tablice „word1” i „word2”.



Zapisując w nowych plikach kolejno grupę, numer litery słowa, czas startu grupy i czas zakończenie wyświetlania grupy, po przecinkach. Czas zakończenia wyświetlania grupy (*time_of_end)*, dopisaliśmy za pomocą formuł excela.

Wczytaliśmy do programu również dane z elektrod (dane_01-12_10_20.txt) jako tablice df. Przejrzeliśmy sygnał na wykresach, a następnie utworzyliśmy tablice „litery” z naszymi czterema łańcuchami liter i zmienną czas0 z czasem startu zapisywania sygnału.



Stworzyliśmy „tablice1” i „tablice2”, które zawierają moment wystąpienia sygnału z wartością mniejszą niż -36500. „tablica1” do pierwszego słowa i „tablica2” do drugiego słowa (gdzie wyszło odpowiednio 16 i 12 wyników).

Utworzyliśmy więc tablice „dana_litera1” i „dana_litera2”, w których zamieniliśmy ponownie nasze dane do bardziej użytkowej formy. Zapisaliśmy zamiast grupy wszystkie litery jako nowe elementy z tablicy, dodaliśmy czas, kiedy dana litera się zaczynała i kończyła wyświetlać oraz która to jest litera z pierwszego i drugiego zakodowanego słowa (numerując od 0). Czyli słowo „owsianka” miało numerację od 0 do 7 (osiem liter), a słowo „Japonia” od 0 do 6 (siedem liter).

```
In [28]: dana_litera1

Out[28]: [['o', 56.150476694107056, 57.150476694107056, 0],
          ['p', 57.150476694107056, 58.150476694107056, 0],
          ['q', 58.150476694107056, 59.150476694107056, 0],
          ['r', 59.150476694107056, 60.150476694107056, 0],
          ['s', 60.150476694107056, 61.150476694107056, 0],
          ['t', 61.150476694107056, 62.150476694107056, 0],
          ['u', 65.25396919250488, 66.25396919250488, 1],
          ['v', 66.25396919250488, 67.25396919250488, 1],
          ['w', 67.25396919250488, 68.25396919250488, 1],
          ['x', 68.25396919250488, 69.25396919250488, 1],
          ['y', 69.25396919250488, 70.25396919250488, 1],
          ['z', 70.25396919250488, 71.25396919250488, 1],
          ['o', 72.90995597839355, 73.90995597839355, 2],
          ['p', 73.90995597839355, 74.90995597839355, 2],
```

Ostatecznie stworzyliśmy tablice „finalne_słowo1” i „finalneP_słowo2”, za pomocą pętli, która wypisała z „tablica1” i „tablica2” odpowiednie wyniki z pomocą tablic „dana_litera1” i „dana_litera2”. Program nasz usunął wszystkie następne wyniki oprócz pierwszego należącego do danego ciągu liter, ponieważ tak jak ustaliśmy wcześniej tylko pierwsze mrugnięcie po wybraniu grupy się liczy. To pozwoliło nam usunąć podwójne mrugnięcia na tej samej literze i przypadkowe mrugnięcia po wybraniu odpowiedniej litery. Zredukowało to mrugnięcia z 16 do poprawnych 8 oraz z 12 do poprawnych 7.

```
In [37]: finalne_slowo1 = []
         y = -1

         for czas in tablica1:
             for przedzial in dana_litera1:
                 if czas < przedzial[2] and czas > przedzial[1]:
                     x = przedzial[3]
                     if x != y:
                         finalne_slowo1.append(przedzial[0])
                         y = x

In [38]: finalne_slowo1

Out[38]: ['o', 'w', 's', 'i', 'a', 'n', 'k', 'a']

In [39]: finalne_slowo2 = []
         y = -1

         for czas in tablica2:
             for przedzial in dana_litera2:
                 if czas < przedzial[2] and czas > przedzial[1]:
                     x = przedzial[3]
                     if x != y:
                         finalne_slowo2.append(przedzial[0])
                         y = x

In [40]: finalne_slowo2

Out[40]: ['j', 'a', 'p', 'o', 'n', 'i', 'a']
```

Jak widać programowi udało się rozkodować nasze dwa wybrane słowa, które, gdyby była taka potrzeba, można by zamienić z tablic na łańcuchy znaków.

4. Opis błędów które się pojawiły:

Zastosowaliśmy zmianę czasu sygnału z bezwzględnego (liczonego od 1.01.1900), na względy liczony od początku uruchomienia programu, w celu łatwiejszego wychwytywania błędów i rozumienia wykresów. Pojawił nam się problem, że momenty, w których było mrugniecie według danych, nie odpowiadały tym na wykresie. Okazało się (nie wiemy z czego to wynika), że ostatni element tablicy t, który miał odpowiadać czasowi bezwzględnemu od początku programu, a ostatni element zbioru danych z kolumny *time*, różni się o 3,5 sekundy, co powoduje przesunięcie wykresu, względem tego, kiedy mrugnięcia są w rzeczywistości. Celem korekcji tego i wyświetleniem rzeczywistego wykresu, jako zmiennej odciętej nie użyliśmy *t* a *df[‘time’]*. Wykres nie przechodził wtedy tak płynnie między kolejnymi punktami natomiast bliżej odpowiadał rzeczywistości i pozwalał znaleźć prawdziwy moment mrugnięcia.

```
In [38]: t[-1],df['time'][len(df['time'])-1]-df['time'][0]

Out[38]: (516.415, 512.9492800235748)
```

Dużym napotkanym kłopotem było niewykrywanie literki *p* w drugim słowie *Ja**p**onia*. Zajęło to sporo czasu, ale okazało się, że ekstremum minimalne po mrugnięciu było wyjątkowo słabe i nie łapało się pod warunek *if df[‘signal’] < - 40000*, ponieważ wynosiło zaledwie -37k. Wystarczyło obniżyć ten próg. Obniżenie to nie spowodowało wyłapania żadnych innych sygnałów. Innym rozwiązaniem byłoby zrobienie bardziej rygorystycznego weryfikowania mrugnięć tzn. warunków odnośnie do ekstremum maksymalnego przed i po minimalnym przy okazji obniżając próg tego minimalnego. Więcej o tym napisaliśmy w możliwych ulepszeniach.

Innym napotkanym problemem było wykrywanie fałszywych mrugnięć tzn. warunki *if df['signal'][i] < - 40000* oraz *if df['signal'][i-1] > -40000* były spełnione, natomiast po kilkunastu milisekundach nasz sygnał ponownie wchodził powyżej -40k a po chwili znowu spadał poniżej -40k. Związane jest to z nieliniowością sygnału i tego, że nie jest on perfekcyjny. W celu odseparowania takich fałszywych mrugnięć wystarczyło dodać warunek, że aby mrugniecie mogłoby być zaliczone musi minąć między nimi co najmniej 200 milisekund.

Następny napotkany problem był łatwy w rozwiązaniu. Gdy na podstawie naszego warunku klasyfikowaliśmy mrugniecie, okazało się ze warunek ten był spełniony, tzn., że ekstremum minimalne występowało około 75ms po samym początku mrugnięcia. Wystarczyło odjąć tą wartość i było wszystko dobrze.

5. Opis możliwych ulepszeń:

Program „speller" i sposób zapisywania danych:

Program do wyświetlania liter na pewno mógłby być jeszcze bliżej dzisiejszych standardów graficzny, do tego pewnie warto by było zmienić bibliotekę graficzną. Sam program mógłby być również bardziej zautomatyzowany. W najprostszej naszej wersji, jakimś ulepszeniem mógłby być przycisk, kiedy kodujący wybiera następne słowo („Następne słowo") oraz program mógł sam zapisywać do pliku txt dane w formacie csv, a nawet takie dane jakie utworzyliśmy w tablicach dana_litera1 i 2.

Analiza danych i rozszyfrowywanie:

Głównym ulepszeniem, które należałoby wprowadzić gdybyśmy mieli więcej słów, to aby każde słowo zapisywało się w tabeli w innej tabeli, a nie w osobnych tabelach jak ma to obecnie miejsce. Zdecydowałem się na ten sposób, mimo tego, że nie miałby zastosowania dla większej liczby słów, dlatego, że łatwiej znaleźć błąd, ponieważ z góry wiemy, którego słowa dotyczy nasz problem.

W przypadku większej liczby słów, należałoby pozmieniać pętle i if'y. Tak aby nie dzieliły zawsze na 2 przypadki (tablica1, word1/tablica2, word2), tylko prawdopodobnie zawrzeć to w kilku większych pętlach, które by dopasowywały kolejne słowa do kolejnych tabel, zagnieżdżonych w jednej głównej tabeli.

Kolejnym pomysłem byłaby dodanie do warunku wykrywania ekstremum minimalnego mrugnięcia, wykrywanie również dwóch ekstremów maksymalnych, które są przed i po ekstremum maksymalnym. Poza sztywną granicą ponad/poniżej które byłyby wykrywane te mrugnięcia, dodatkowo można by wprowadzić stosunek proporcji między *wielkościami* tych ekstremów od osi odciętej, która w tym przypadku to 0 V.

Synchronizacja:

Ostatecznym ulepszeniem mogłoby być zaprojektowanie programu, który w czasie rzeczywistym zapisywałby litery, które odmrugiwane są przez kodującego. W tym celu byśmy musieli połączyć nasze dwa programy (i pewnie wprowadzić szereg poprawek).