Symulator maszyny szyfrującej Enigma

Projekt na przedmiot Podstawy Informatyki i Programowania, semestr 21Z

Autor: Kajetan Rożej

Wstęp

Enigma to niemiecka przenośna elektromechaniczna maszyna szyfrująca, oparta na mechanizmie obracających się wirników, skonstruowana przez Artura Scherbiusa. Chociaż w użytku komercyjnym znajdowała się już od lat 20 XX wieku największą „sławę" zawdzięcza II Wojnie Światowej podczas której wykorzystywana była przez stronę niemiecką do kodowania wiadomości wojskowych. Ocenia się, że złamanie szyfru Enigmy przez Aliantów pozwoliło zakończyć wojnę kilka lat wcześniej. W tym przełomowym wydarzeniu niebagatelną rolę odegrali również polscy kryptolodzy: Marian Rejewski, Jerzy Różycki i Henryk Zygalski.

Cel Projektu

Celem projektu było napisanie programu komputerowego który swym działaniem symulowałby maszynę szyfrującą Enigma. Nie chodziło jednak o odwzorowywanie konkretnego modelu urządzenia, a o uogólnioną implementację maszyny wirnikowej, z niezdefiniowaną odgórnie liczbą wirników (w przeciwieństwie do fizycznych urządzeń). Stworzony program miał mieć możliwość kodowania tekstu litera po literze (jak w oryginalnej maszynie) oraz kodowania tekstu z pliku. Dodatkową wartością symulatora miał być element edukacyjno-poznawczy, pozwalający prześledzić „drogę" kodowanej litery w celu lepszego zrozumienia działania maszyny.

Wykorzystane narzędzia

Projekt zrealizowano w języku programowania Python, wersja 3.8.10. Interfejs graficzny napisano z wykorzystaniem biblioteki Qt i Pyside. Do edycji kodu wykorzytsano program Visual Studio Code. Program testowano z użyciem frameworka pytest w wersji 6.2.5

Struktura Projektu

```
.
|_ enigma_classes
|   |_ rotor_class.py
|   |_ plugboard_class.py
|   |_ reflector_class.py
|   |_ enigma_class.py
|   |_ functions.py
|
|_ enigma_gui
|   |_ gui.py
|   |_ main_window.py
|   |_ enigma.ui
|   |_ table_models.py
|   |_ dialogs.py
|       |_ errors.py
|
|_    rsc
|   |_ config.json
|   |_ default.json
|   |_ custom. json
|
|_   tests
|   |_ test_rotor_class.py
|   |_ test_plugboard_class.py
|   |_ test_reflector_class.py
|   |_ test_enigma_class.py
|   |_ test_functions.py
|   |_ test_rsc_manager.py
|   |_ test_elements_database.py
|
|_ rsc_manager.py
|_ elements_database.py
|_ enigma.py
|_ README.md
```

Na strukturę projektu składają się cztery podkatalogi:

- enigma_classes - z implementacją klas poszczególnych elementów Enigmy i samej maszyny:

    - rotor_class.py – implementacja klasy reprezentującej wirinik
    - plugboard_class.py – implementacja klasy reprezentującej łącznicę kablową
    - reflector_class.py – implementacja klasy reprezentującej bęben odbijający
    - enigma_class.py – implementacja klasy reprezentującej całą maszynę z wirnikami, bębnem odbijającym i łącznicą kablową
    - functions.py – wyodrębnienie funkcji wykorzystywanych w wielu klasach

- enigma_gui – zawierające pliki z implementacją interfejsu graficznego:

    - gui.py - główny plik z klasą EnigmaWindow (zasadnicze okno programu)
    - main_window.py - kompozycja głównego okna programu wygenerowana z pliku .ui przy użyciu pyside-uic
    - enigma.ui - kompozycja głównego okna stworzona przy użyciu designera
    - table_models.py - modele tabel wyświatlanych w interfejsie
    - dialogs.py - definicje okien dialogowych wyświetlanych w ramach interfejsu
    - errors.py - definicje informacji o błędach wyświetlanych w ramach interfejsu

- rsc - zawierający pliki z zasobami i zapisaną konfigurację:

    - config.json - domyślna konfiguracja maszyny szyfrującej
    - default.json - baza domyślnych elementów (wirników i bębnów odbijających) pochodzących z oryginalnej Enigmy
    - custom.json - baza elementów zdefiniowanych przez użytkownika

- tests - zawierający testy jednostkowe do projektu

oraz cztery pliki w katalogu głównym:

- rsc_manager.py - odpowiadający za zarządzanie plikami (zapis i odczyt oraz podstawowa obróbka) znajdującymi się w katalogu w katalogu rsc
- elements_database.py - zawierająca obiekt bazy danych dostępnych obiektów
- enigma.py - główny program, wywoływany z konsoli z odpowiednimi parametrami
- README.md - plik z dokumentacją

Format plików konfiguracyjncyh

- config.json :

    - machine - parametry istotne przy inicjalizacji maszyny *(dict)* z najstepującymi kluczami:
        - rotors - lista nazw wirników *list(str)* w kolejności w jakiej zostaną umieszczone w maszynie. Nazwa musi zgadzać się z nazwą w bazie elementów.
        - rings - duże litery angielskiego alfabetu w ilości odpowiadającej ilości wirników *(str)* - kolejne litery oznaczają pozycje pierścieni na kolejnych wirnikach
        - start_position - duże litery angielskiego alfabetu w ilości odpowiadającej ilości wirników *(str)* - kolejne litery oznaczają pozycje początkowe kolejnych wirników
        - reflector - nazwa bębna odbijającego *(str)* do zamontowania w maszynie. Nazwa musi zgadzać się z nazwą w bazie elementów.
        - plugboard - pary wielkich angielskich liter (bez powtózeń) rozdzielone spacjami *(str)* symbolizujące łącznice kablową (np. 'AB CD' oznacza, że A jest połączone z B a C z D)
    - settings - dodatkowe ustawnia *(dict)* z najstepującymi kluczami
        - double_step - flaga *(bool)* informująca, czy maszyna ma wykonywać podwójny krok, jeśli będzie w odpoweidniej pozycji (ma znaczenia tylko, gdy w maszynie zamontowane są 3 wirniki)
        - space_dist - odstęp między spacjami *(int)* w kodowanym tekście - 0 oznacza tekst bez spacji

```
{
    "machine": {
        "rotors": [
        ],
        "rings": "QAA",
        "start_positions": "TAA",
        "reflector": "reflectorUKWC",
        "plugboard": "AS DF RY"
    },
    "settings": {
        "double_step": true
        "space_dist": 5
    }
}
```

- custom.json and default.json:

    - rotors - lista *(list)* wirników *(dict)* z najstepującymi kluczami:

        - name - unikalna nazwa własna wirnika *(str)*
        - wiring - okablowanie wirnika składające się ze wszystkich wielkich liter alfabetu angielskiego *(str)* bez spacji (np. 'DF...Q' oznacza, że w podstawowej pozycji wirnika A odpowiada D, B-F, a Z - Q)
        - indentation - pozycja lub pozycje (max 2) z wcięciami - jedna lub dwie różne wielkie litery angielksiego alfabetu *(str)*

    - reflectors - lista *(list)* bębnów odbijających *(dict)* z najstepującymi kluczami:

        - name - unikalna nazwa własna bębna odbijającego *(str)*
        - wiring - okablowanie bębna odbijajacego składające się ze wszystkich wielkich liter alfabetu angielskiego *(str)* pogrupowanych w pary rozdzielone spacją (np. 'DF AS ...' oznacza, że D jest kodowane na F, a A na S i odwrotnie)

```
{
    "rotors": [

        {
            "name": "ROTOR1",
            "wiring": "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
            "indentations": "AF"
        },

        {
            "name": "ROTOE1",
            "wiring": "BCDEFGHIJKLMNOPQRSTUVWXYZA",
            "indentations": "T"
        }
    ],
    "reflectors": [

        {
            "name": "reflectorUKWB",
            "wiring": "AY BR CU DH EQ FS GL IP JX KN MO TZ VW"
        }
    ]
}
```

Nie jest zalecane samodzielne modyfikowanie plików konfiguracyjnych!!

Format pliku wejściowego

Plik wejściowy z tekstem do zakodowania nie ma sprecyzowanego formatu. Może jednak zawierać tylko wielkie litery angielskiego alfabetu (niedopuszczalne są m. in. spacje). Tekst może być zapisany w wielu liniach

Instrukcja użytkownika

Uruchomienia programu należy dokonać poprzez wywołanie pliku enigma.py z konsoli z odpowiednimi parametrami:

```
usage: enigma.py [-h] [-m] [-i] [-o] [-c]

Simulate Enigma machine.

optional arguments:
-h, --help           show this help message and exit
-m , --mode          define the mode in wich programme should be run [gui/cmd] (default: gui)
-i , --input_file    file with plain text to encode, required in cmd mode
-o , --output_file   file with result (default: result.txt)
-c , --config        json file with initial configuration of machine (default: .../rsc/config.json)
```

Domyślnie (bez podania żadnych parametrów) program uruchomi się w postaci interfejsu graficznego.

Tryb interfejsu graficznego (-m gui)

Funkcje:

- związane stricte z maszyną szyfrującą
    - kodowanie tekstu litera po literze wraz z wyświetleniem poszczególnych kroków
    - kodowanie tekstu z pliku wejściowego do pliku wynikowego
    - zmiana obecnej konfiguracji maszyny
- związane ogólnie z działaniem symulatora
    - dodanie/usunięcie/modyfikacja spersonalizowanych wirników
    - dodanie/usunięcie/modyfikacja spersonalizowanych bębnów odbijających
    - zmiana ustawień

Kodowanie litera po literze

- *Letter to encrypt* - JEDNA duża litera alfabetu angielskiego do zakodowania
- *Encrypt* - koduje literę podaną w polu *Letter to encrypt*

Kodowanie z pliku

- *Browse (input file)* - otwiera okno do wyboru pliku do zakodowania
- *Browse (output file)* - otwiera okno do wyboru pliku wynikowego

- *Encrypt* - koduje tekst z pliku wejściowego do pliku wynikowego w obecnej konfiguracji

Zmiana obecnej konfiguracji

- Przycisk *+* - otwiera okno dodania wirnika do maszyny (z bazy elementów). Po dodaniu rotor zostaje dodany na koniec listy wirnika
- *Load configuration from file* - otwiera okno do wyboru pliku z konfiguracją. Format musi być zgodny z formatem pliku config.json
- *Restore default configuration* - przywraca domyślną konfigurację z pliku config.json UWAGA! - przywraca również domyślne ustawienia w settings
- *Reflector* - możliwość wyboru bębna odbijającego maszyny (z bazy elementów)
- Plugboard *Set* - ustawia podaną w polu tekstowym łącznicę kablową (jeśli format jest poprawny)
- *Save as default* - zapisuaje obecną konfigurację jako domyślną (do pliku config.json)
- *Export* - otwiera okno do wyboru pliku, do którgo obecne ustawienia mają zostać wyeksportowane

Bieżąca konfiguracja jest zmieniana na bieżąco i nie ma potrzeby jej zapisywać przed przełączeniem karty Domyślna konfoguracja jest zmieniana tylko po kliknięciu w przycisk *Save as default*

Dostepne po wybraniu wirnika z listy:

- Przycisk -- usuwa wybrany wirnik z listy
- Przycisk /\ - przesuwa wybrany wirnik o jedną pozycję w górę listy (jeśli wirnik jest na pierwszej pozycji - nic się nie dzieje)
- Przycisk \/ - przesuwa wybrany wirnik o jedną pozycję w dół listy (jeśli wirnik jest na ostatniej pozycji - nic się nie dzieje)
- *Position* - możliwość ustawienia pozcyji wirnika
- *Ring* - możliwość ustawienia pierścienia wirnika

Dodanie/usunięcie/modyfikacja spersonalizowanych wirników

- Przycisk *+* -
    - *Name* - nazwa nowego wirnika - nie może być pusta i musi być unikatowa
    - *Wiring* - okabblowanie nowego wirnika - wszytskie wielkie litery angielskiego alfabetu bez spacji np. 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    - *Indentation(s)* - wcięcie nowego wirnika - jedna lub dwie wielkie litery angielskiego alfabaetu bez spacji np. 'AF'
    - *Cancel* - anuluje dodawanie wirnika
    - *Add* - dodaje wirnik jeśli wszytskie pola spełniają wymagania
- *Add from file* - otwiera okno dialogowe do wyboru pliku z bazą danych do załadowania (musi być w formacie takim jak custom.json). UWAGA: Ładuje zarówno wirniki jak i bębny, więc w żadnej z baz nie może być konfliktu

Po wybraniu wirnika:

- Przycisk -- usuwa wirnik z bazy jeśli aktualnie nie jest w użyciu (nie jest w bieżącej ani domyślnej konfiguracji)
- *Name* - nowa nazwa wirnika - nie może być pusta i musi być unikatowa
- *Wiring* - nowe okabblowanie wirnika - wszytskie wielkie litery angielskiego alfabetu bez spacji np. 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
- *Indentation(s)* - nowe wcięcie wirnika - jedna lub dwie wielkie litery angielskiego alfabaetu bez spacji np. 'AF'
- *Modify* - modyfikuje wirnik zgodnie z powyższymi polami jeśli wszytskie pola spełniają wymagania

Dodanie/usunięcie/modyfikacja spersonalizowanych bębnów odbijających

- Przycisk *+* -
    - *Name* - nazwa nowego bębna - nie może być pusta i musi być unikatowa
    - *Wiring* - okablowanie bebna - wsyztskie wielkie litery angielskiego alfabaetu pogrupowane w pary rozdzielone spacją np. 'AB CD EF GH IJ KL MN OP QR ST UV WX YZ'
    - *Cancel* - anuluje dodwanie bębna
    - *Add* - dodaje bęben jeśli wszytskie pola spełniają wymagania
- *Add from file* - otwiera okno dialogowe do wyboru pliku z bazą danych do załadowania (musi być w formacie takim jak custom.json) UWAGA: Ładuje zarówno wirniki jak i bębny, więc w żadnej z baz nie może być konfliktu

Po wybraniu bębna:

- Przycisk -- usuwa bęben z bazy jeśli aktualnie nie jest w użyciu (nie jest w bieżącej ani domyślnej konfiguracji)
- *Name* - nowa nazwa istniejącego bębna - nie może być pusta i musi być unikatowa
- *Wiring* - okablowanie bebna - wsyztskie wielkie litery angielskiego alfabaetu pogrupowane w pary rozdzielone spacją np. 'AB CD EF GH IJ KL MN OP QR ST UV WX YZ'
- *Modify* - modyfikuje bęben zgodnie z powyższymi polami jeśli wszytskie pola spełniają wymagania

Ustawienia

- *Double step* - zmienia aktywność podwójnego kroku - dostępne tylko przy 3 wirnikach w maszynie
- *Space after ... signs* - ustala liczbę liter pomiędzy spacjami (jeśli 0 to tekst jest generowany bez spacji). Jeśli nowa wartość jest większa niż ilość już zakodowanych liter spacja jest dodawana przy pierwszym kodowaniu
- *Save as default* - zapisuje bieżące ustawienia jako domyślne

Bieżące ustawienia są zmieniane na bieżąco i nie ma potrzeby ich zapisywać przed przełączeniem karty domyślne ustawienia są zmieniane tylko po kliknięciu w przycisk *Save as default*

Każdej z funkcji odpowiada jedno okno programu, do którego przenieść się można dzięki rozwijanemu górnemu menu

Tryb konsolowy (-m cmd)

W trybie konsolowym program dla pliku wejściowego (do którego ścieżka podana jest jako parametr -i albo --input_file) generuje plik wynikowy (w lokalizacji podanej jako parametr -o albo --output_file lub jako result.txt). Jeśli podana została ścieżka do pliku konfiguracyjnego (jako paramater -c albo --config) to plik zakodowany zostanie dla podanej konfiguracji początkowej (format musi być zgodny z formatem dla pliku config.json). Jeśli ścieżka taka nie została podana, plik kodowany jest z konfiguracją zapisaną w ./rsc/config.json. Jeśli ścieżka do pliku wejściowego nie została podana wykonanie programu nie powiedzie się.

Podsumowanie

Projekt w ostatecznym kształcie posiada pełną założoną funkcjonalność. Podczas jego tworzenia udało się dodać pewne nieplanowane wcześniej

usprawnienia/ułatwienia, które nie były skomplikowane w implementacji a znacząco ułatwiły korzystanie z niektórych funkcji - np. pasek z podświetleniem wykorzystanych liter przy polach tekstowych, czy użycie QFileDialog do przeglądania plików. Dzięki nim korzytsanie z symulatora przy użyciu GUI jest wygodne i intuicyjne. Wpływ na to ma również rozbudowany system obsługi wyjątków, które wyświatlane są użytkownikowi w postacji jasnych komunikatów. Oczywiście projekt posiada potencjał do dalszego ewentualnego rozwoju - w przyszłości rozważyć możnaby było dodanie polskiej wersji językowej, przestawienie połączeń łacznicy kablowej w postacji graficznej, czy możliwość dodania historii zakodownych plkików.