The Enigma Machine Simulator

Patryk J. Będkowski

Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Styczeń 2021

Niniejszy projekt został utworzony w ramach przedmiotu Podstawy Informatyki i Programowania.

Spis Treści

I. Opis Projektu

Symulator Maszyny Szyfrującej Enigma jest programem, który symuluje działanie maszyny szyfrującej wykorzystywanej podczas Drugiej Wojny Światowej.

II. Wymagania Uruchomieniowe

Prawidłowe działanie programu wymaga zainstalowania **interpretera Python w wersji co najmniej 3.6.**

Wymagane jest również zainstalowanie modułów używając komend:

```
pip -install pyqt5
pip -install tabulate
pip -install pyfiglet
```

bądź poprzez uruchomienie pliku `requirements.txt,` używając komendy:

```
pip install -r requirements.txt
```

Zaleca się utworzenie oddzielnego środowiska uruchomieniowego python.

III. Uruchomienie programu

W zależności od wyboru trybu pracy, program można uruchomić w dalej opisany sposób.
Symulator rozpoczyna pracę po wpisaniu w wierszu poleceń następującej komendy:

```
python enigma.py
```

Aby uruchomić program w trypie interfejsu graficznego, należy wprowadzić komendę:

```
python enigma_gui.py
```

IV. Opis Modułów

1. enigma_class

Zawiera implementację klasy `Enigma`, której metody stanowią główną część algorytmu przetwarzania tekstu. Wartości wstawiane do tej klasy są sprawdzane pod kątem ich poprawności, które są przedstawione w sekcji **nr. V**. Jeżeli wprowadzona przez użytkownika wartość nie spełnia założeń programu, wyświetlony zostaje wyjątek ze stosownym komunikatem. Sprawdzeniu poprawności danych podawane są również nieprawidłowo skonfigurowane pliki ustawień .json.

1.1. Enigma()

Obiekt klasy Enigma stanowi główną część programu. Przechowuje informacje o wprowadzonych ustawieniach. Posiada metody umożliwiające zapis wprowadzonych ustawień oraz sprawdzenia poprawności wprowadzonych danych.

Atrybut `list_of_rotors` musi posiadać typ `list`, musi również składać się z trzech elementów, które są typami `int` oraz każdy z nich musi być w przedziale od 1 do 26. Ten warunek jest sprawdzany przez metodę `check_set_rotor_value().`

Atrybut `steckerbrett` musi posiadać typ `dict,` nie może przechowywać dwóch powtarzających się elementów (tzn. kluczy bądź wartości), przechowywane elementy muszą należeć do alfabetu `ascii.`Te warunki są sprawdzane przez metody `steckerbrett_check_for_same_values()` oraz `steckerbrett_check_values().`

Atrybut `reflector` musi mieć typ `str`, oraz być jedną z wartości `A, B, C`. Ten warunek jest sprawdzany przez metodę `reflector_check_model().`

Szyfrowanie tekstu odbywa się po wywołaniu modułu `encryptingCodec` podając szyfrowany tekst jako jej parametr.

2. enigma_interface

Moduł zawierający implementację klasy interfejsu `Enigma_interface()` odpowiadającej za interfejs użytkownika. Do ważniejszych metod należą:

1.1. design_assumptions()

Metoda klasy interfejsu zwracająca założenia projektowe wprowadzanych wartości i ustawień. Każda pozycja przedstawia reguły uruchamianych ustawień niezbędnych do bezproblemowego działania symulatora.

1.2. main_menu()

Metoda obsługująca kolejne etapy działania interfejsu. Gdy podczas jej wykonywania nastąpi przerwanie pracy poprzez skrót klawiszowy `Ctrl + c` zostanie wyświetlony stosowny komunikat.

1.3. start_menu()

Metoda wyświetlająca użytkownikowi dostępne opcje programu. Na tym etapie występuje możliwość:

- [1] Odczyt wprowadzanego tekstu z pliku .txt
- [2] Wprowadzenie własnego tekstu z klawiatury
- [3] Wyświetlenia założeń projektowych
- [4] Zamknięcie programu

Wybierając jedną z opcji `1,2,3` użytkownik ma możliwość powrotu do głównego MENU, wpisując opcję `y.`

Po wprowadzeniu numeru opcji **1** przez użytkownika, następuje odwołanie się do metody `input_txt_file(),` w której użytkownik wpisuje ścieżkę do wprowadzanego pliku. Jeżeli plik zostaje odczytany, po wywołaniu funkcji `read_txt_file()` znajdującej się w warstwie trwałości, następuje przypisanie do atrybutu klasy tekstu do przetworzenia.

Jeżeli została wprowadzona opcja **2** przez użytkownika, następuje odwołanie się do metody `input_txt_file(),` w której następnie tekst zostaje wpisany z klawiatury i zapisany w atrybucie klasy.

W obu przypadkach, kolejno następuje przerwanie pętli i przejście do metody `setting_menu()`

1.4. setting_menu()

Użytkownik jest proszony o wybór sposobu wprowadzenia ustawień do symulatora. Wynik ten zostaje przypisany do odpowiedniego atrybutu klasy, aby mógł być później wykorzystany przy eksportowaniu ustawień.

Wybierając **y**, program przechodzi do metody `import_settings_from_file()`. Użytkownik wprowadza ścieżkę do pliku .json a następnie zostaje uruchomiona funkcja read_json_file(), która zwraca trzy ustawienia.

Wybierając **n**, program przechodzi do metody `insert_settings_by_hand()`. Użytkownik wprowadza po kolei ustawienia z klawiatury. Na tym etapie następuje sprawdzenie danych wejściowych nie co do ich wartości, lecz co do poprawności wprowadzanego formatowania (np. ilość wprowadzonych ustawień, puste, powtórzone wartości, złe formatowanie).

W obu przypadkach metody pobierają trzy ustawienia:

- list_of_rotors
- steckerbrett
- reflector

W kolejnym korku następuje wywołanie metody `initiate_enigma_simulator()`, wraz z ustawieniami jako jej argumenty.

1.5. initiate_enigma_simulator()

Główna metoda, w której następuje proces odwołania się do klasy `Enigma` i wywołanie przetworzonego tekstu.

W przypadku wprowadzenia ustawień, które powodują wywołanie wyjątku, następuje ukazanie się odpowiedniego komunikatu użytkownikowi. Użytkownik jest również proszony o ponowne wprowadzenie ustawień, jeżeli są błędne. Następuje wywołanie metody `initiate_enigma_again()`.

Jeżeli wprowadzone dane zostają zaakceptowane i program prawidłowo zwraca przetworzony tekst, następuje wywołanie metody `export_txt_menu()`.

1.6. export_txt_menu().

Użytkownik ma możliwość wyrażenia chęci utworzenia pliku .txt z przetworzoną wiadomością. Jeżeli wprowadzi y następuje przejście do metody `export_txt_file()` oraz wywołanie funkcji `save_txt_file()`, w której użytkownik podaje nazwę pliku (spełniającą założenia), który zostaje utworzony w głównym katalogu z programem.

Zostaje przerwana pętla `while` za pomocą `break` i następuje wywołanie powrót do metody `initiate_enigma_simulator()`.

Ostatnim krokiem programu jest sprawdzenie czy użytkownik zaimportował plik z ustawieniami. Jeżeli to zrobił, prawdopodobnie nie potrzebuje tworzyć go jeszcze raz. Następuje wyświetlenie stosownego komunikatu i przerwanie działania programu.

Jeżeli użytkownik nie wprowadził ustawień z pliku (tylko ręcznie z klawiatury), następuje wywołanie metody `export_json_menu().`

1.7. export_json_menu().

Jeżeli użytkownik na początku działania programu zdecydował się wprowadzić ustawienia *ręcznie* , ma możliwość zapisania tych ustawień do pliku `.json.` W przeciwnym wypadku nie zostaje wyświetlone zapytanie o utworzeniu pliku z ustawieniami. Jeżeli zdecyduje się na zapis ustawień do pliku wpisując opcję **`y,`** następuje odwołanie do metody `export_json_menu().` Program kończy działanie wyświetlając stosowny komunikat.

3. file_management

Warstwa trwałości symulatora, obsługująca operacje odczytu i zapisu do pliku. Złożona jest z pięciu funkcji:

1.1. check_if_ascii()

sprawdza czy wartość jej argumentu znajduje się w alfabecie ascii.

1.2. read_txt_file()

jako argument przyjmuje ścieżkę do pliku tekstowego. Otwiera ten plik i pobiera z niego zawartość. Rzuca wyjątek jeżeli plik nie został odnaleziony, plik jest pusty, ilość wypełnionych linii jest różna od 1, znak/znaki w nim zawarte nie należą do alfabetu ascii.

1.3. save_txt_file()

jako argument przyjmuje przetworzony tekst. Plik posiada nazwę wprowadzoną przez użytkownika za pomocą funkcji `input`. Rzuca wyjątek jeżeli nazwa nie została podana, nazwa pliku zawiera znaki inne od znaków alfabetu ascii oraz cyfr rzeczywistych. Jeżeli nazwa jest prawidłowa, tworzy plik w głównym katalogu.

1.4. read_json_file()

jako argument przyjmuje ścieżkę do pliku z rozszerzeniem .json. Otwiera ten plik i pobiera z niego zawartość. Rzuca wyjątek jeżeli plik nie został odnaleziony. Funkcja zwraca słownik zawierający ustawienia symulatora.

1.5. save_json_file()

jeżeli spełnione są wszystkie założenia, tworzy plik zawierający ustawienia, przyjmowane z argumentu, symulatora w formie słownika. Plik posiada nazwę wprowadzoną przez użytkownika za pomocą funkcji `input`. Rzuca wyjątek jeżeli nazwa nie została podana, nazwa pliku zawiera znaki inne od znaków alfabetu ascii oraz cyfr rzeczywistych.

4. enigma_gui

Moduł uruchomieniowy programu w trybie interfejsu graficznego.

5. gui

Moduł zawierający klasy: `EnigmaUi`, która odpowiedzialna jest za budowę interfejsu, klasę `EnigmaWindow` odpowiedzialną za funkcjonowanie interfejsu graficznego.

6. enigma

Główny moduł uruchomieniowy zawierający funkcję `main()`, za pomocą której następuje uruchomienie symulatora.

7. exceptions

Moduł zawierający wyjątki dotyczące wprowadzanych wartości, sprawdzane w symulatorze.

V. Założenia wprowadzanych danych

Zawiera sformułowania dotyczące poprawności wprowadzanych danych do symulatora.

1.1. Tekst wprowadzany z klawiatury

- musi się składać tylko z liter współczesnego alfabetu łacińskiego zbudowanego z 26 liter
- musi zawierać tylko duże litery
- musi być wprowadzony bez odstępów, znaku spacji

1.2. Steckerbrett (mechanizm łączenia liter w pary)

- musi zostać wprowadzony jako pary dwóch liter oddzielone przecinkiem, przykład: AB,CD,EF,GH
- nie może składać się z dwóch takich samych liter
- musi się składać tylko z dużych liter współczesnego alfabetu łacińskiego
- może nie zawierać żadnych wprowadzeń

1.3. Wirniki (Rotors)

- musi się składać z cyfr całkowitych większych bądź równych 1 oraz mniejszych bądź równych 26
- musi się składać z trzech cyfr oddzielonych dwoma przecinkami, przykład: 1,2,3
- musi być wprowadzony bez odstępów, znaku spacji

1.4. Reflektor (Reflector)

- musi zostać wprowadzony jako jedna z przedstawionych opcji „A", „B" bądź „C"

1.5. Nazwa eksportowanego pliku

- może się składać wyłącznie z liter współczesnego alfabetu łacińskiego bądź cyfr nieujemnych całkowitych
- litery mogą być wprowadzane w dużym bądź małym formacie

1.6. Wprowadzany plik tekstowy

- musi się składać tylko z liter współczesnego alfabetu łacińskiego zbudowanego z 26 liter
- musi zawierać tylko duże litery
- musi być wprowadzony bez odstępów, znaku spacji
- musi składać się tylko z jednej linijki

1.7. Wybory opcji

- muszą zostać wprowadzone spośród przedstawionych przez symulator, w tym samym formacie, przykład: „Insert y/n" – wprowadzona opcja musi być literą „y" (yes) bądź „n" (no)