Morse Code Frequency Graph Application

DSAA CA1

By Shaun Kwo Rui Yu

Adm No: 2317933

Class: DAAA/FT/2A/03

Contents

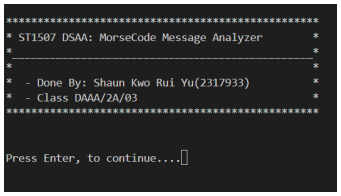
Introduction

The Morse Code Frequency Graph Application is designed to examine Morse code from a textual content file, calculate the frequency of every Morse code key-word, and display this records as a vertical bar graph. This report outlines the application's functionality, presents person guidelines,

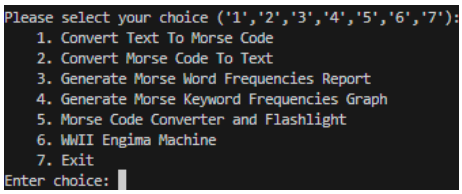
discusses the OOP approach, information the information structures and algorithms used, and summarizes the demanding situations confronted during development.

How to run application

To run this application, you can open with VS Code or open the terminal where python is set as a global environment variable. To run this from code editor open main.py and run it you will have a running application. To run this application from terminal type python main.py. I will start running. This will appear on console. after this press enter button.



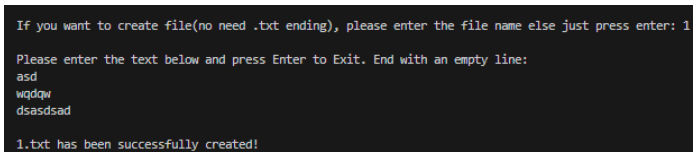
After pressing enter it will lead to option menu and select the options from menu.



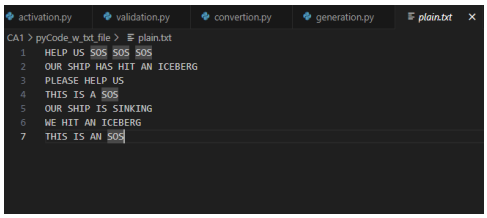
Option 1

Now select the option 1 which will:

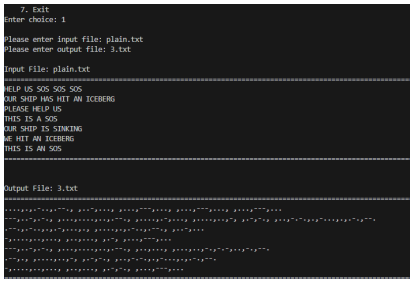
- Prompt user to create a new word file first if no files have been created yet and only the file name without .txt needs to be input as the code will append.txt by itself
- Then, convert words of input text file to Morse code and stores it in output file



Create file if user want to or don't have file



Text file has been created and whenever 'Cancel' is pressed program will skip to ask for input file and output file



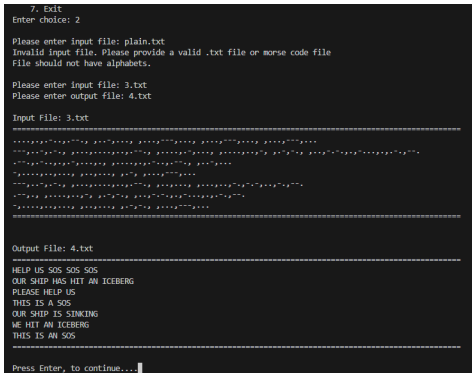
Display Files with some error checking at the top

This is encoding the English to Morse Code

Option 2

Now select the option 2 which will

- Prompt user to create a new morse code file first if no files have been created yet and do as per above instructions as it is the same
- Then, convert Morse code of input file to words and stores it in output file

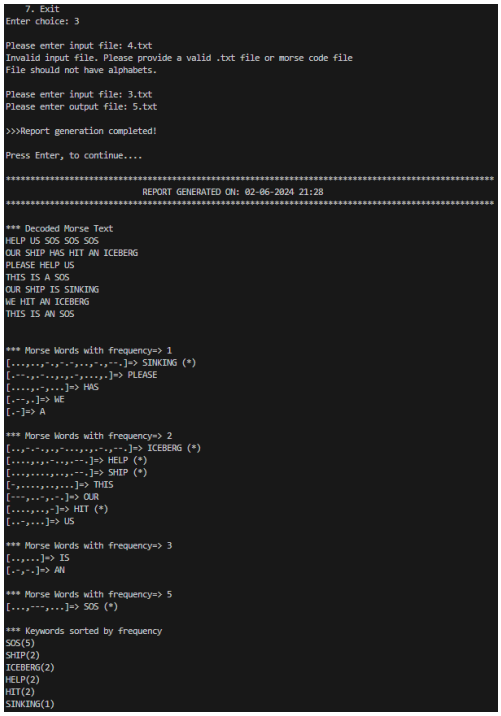


Display Files with some error checking at the top

This is decoding the Morse code to English text.

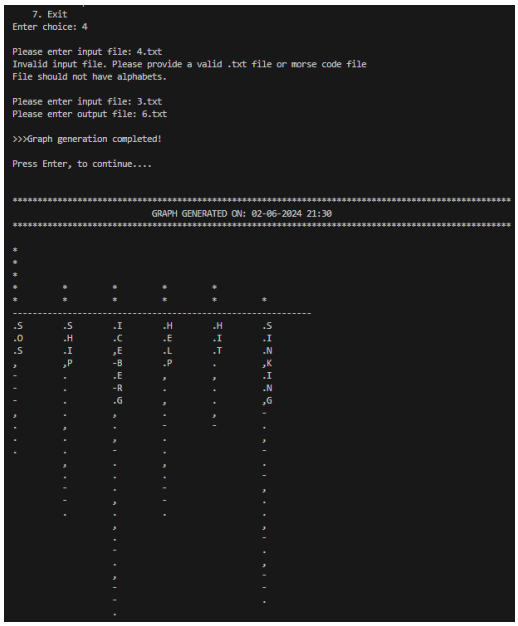
Option 3

Now press 3, put in input and output files, to see the report generated with validation at the top.



Option 4

Now press option 4, put in input and output files, to generate the graphs with validation at the top.

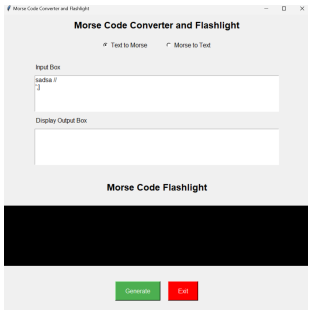


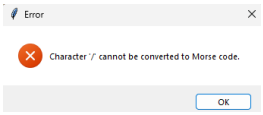
Option 5

For part 5, I have done this additional gui based feature which will help to read long texts and create flashlight (black to yellow) and sound effects (beeps) like how an actual morse code machine works

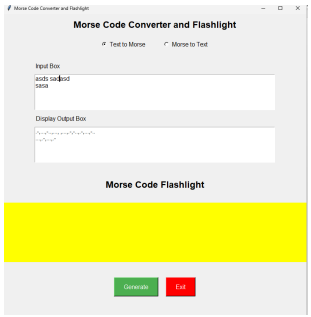
Here is how it will look like when nothing is put into the 1st box on top which is the input, and the result will be show on the output box

When select Text to Morse radio button, here is the error checking for non-alphabetical text,
Click the ‘Generate’ button to test and an error prompt will pop up and no result will be displayed



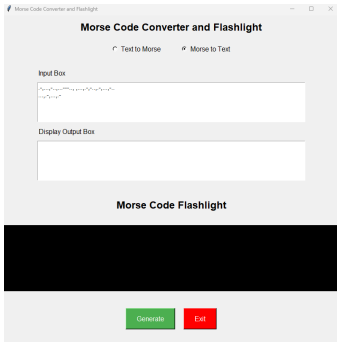


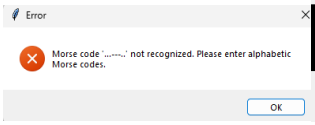
If only alphabetical text is given, click the ‘Generate’ button and the code will work

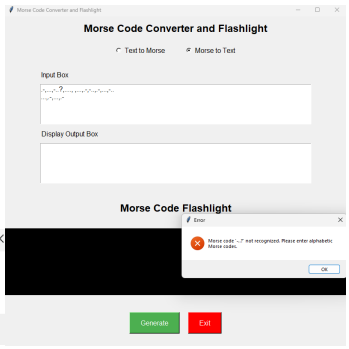


Note: If you press any radio button during the successful code running the code will stop and move on to the other radio button’s function and clear the textboxes and stop the sound.

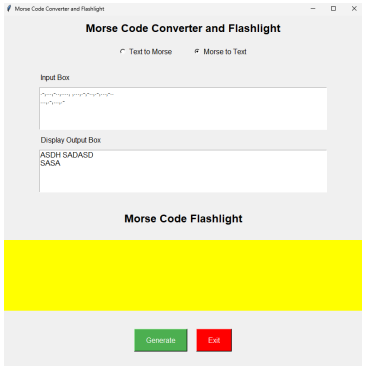
Moving on to Morse to Text function, here is the error output if there is error where no result will be displayed in the Display Output Box





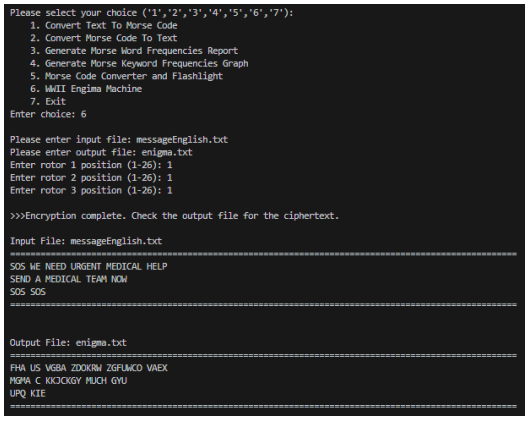


Here is if there is morse code that is alphabetical, click the ‘Generate’ button and the code will work

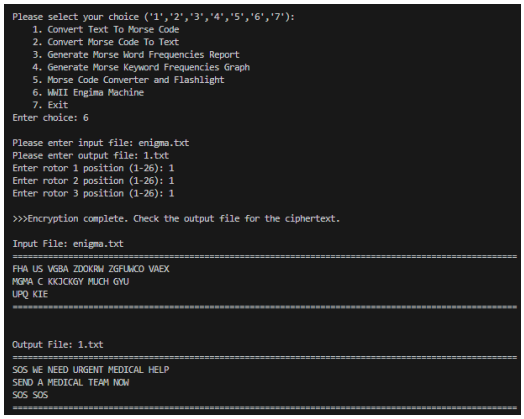


Option 6

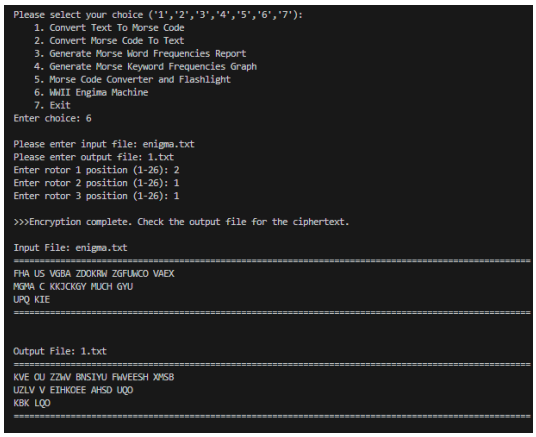
Simplified Engima Machine: Before encoding to Morse Code during WWII, Germany had an interesting way of encoding the message before sending it to mobile platoons on the battlefield, to prevent messages to be read by enemies, they used the enigma machine to change the words. The machine actually had rotors, reflector and a plugboard but for simplicity sake, I am only doing rotors



I set all rotors to 1,1,1 and display the output message which looks like a lot of gibberish, then I try to convert it back. Encoding and decoding is the same but only works with same rotor configuration

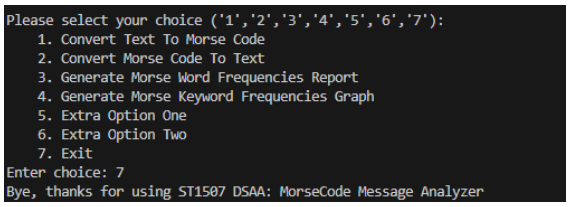


If incorrect rotor configuration is used for example 2,1,1, so even if the enemy knows that there is a machine with 3 rotors for encoding, it will take very long for them to guess the correct configuration



Option 7

Now press 7 to exit



I have designed code as per what is said in the CA1 brief:

- validator Class: Ensures the input file exists and contains valid Morse code.
- converter Class: Handles the conversion of text to Morse code.
- generator Class: Manages the generation of the Morse code frequency graph.
- MorseCodeApp Class: Creates a GUI for user to put text needed to be encoded or decoded
- EngimaMachina Class: Creates a simulation/ example of how a simplified enigma machine works
- Main.py is to run the whole application

Activation class

The `MorseCodeAnalyzer` class integrates essential functionalities for analyzing Morse code messages using a structured approach. The `__init__` method initializes the `validator`, `converter`, and `generator` classes, which handle file validation, Morse code conversion, and report generation. The `main` method provides a user interface with options to convert text to Morse code, convert Morse code to text, generate frequency reports, and create keyword frequency graphs. Key data structures include lists for storing graph lines and Morse keywords, and dictionaries for counting word frequencies. These structures were chosen for their efficiency in handling lookups, updates, and manipulation, ensuring optimal performance. The use of strings simplifies content manipulation and output formatting. This design leverages Python's strengths, making the application efficient, clear, and maintainable.

Conversion class

The `conversion` class in this application provides essential methods for converting between text and Morse code, utilizing efficient data structures for optimal performance. The `__init__` method initializes dictionaries for quick lookups during conversions, mapping characters to Morse code and vice versa. The `encode_morse` method reads plain text from an input file, converts it to Morse code, and writes it to an output file, using lists for temporary storage. Conversely, the `decode_morse` method reads Morse code, decodes it to text, and handles invalid inputs by prompting for a valid file. The `display_text_files` method uses `easygui` to display file contents graphically and on the console for verification. Additionally, the `create_input_file` method allows users to generate input files through an intuitive graphical interface. These methods and data structures—primarily dictionaries and lists—ensure efficient, clear, and maintainable text and Morse code conversions.

Generation class

The `generator` class provides methods to analyze Morse code and generate reports and graphs, leveraging dictionaries and efficient data handling. It initializes with text-to-Morse and Morse-to-text mappings and can read stopwords to filter common words. The `store_output_data` method calculates word frequencies in the input file, excluding stopwords, and organizes words by frequency for reporting. The `generate_report ` method generates a detailed report with Morse and text frequencies. The `transposed_graph` method creates a vertical graph of word frequencies in Morse code, aligning stars and Morse characters for a visual representation. These methods utilize Python's dictionaries and file I/O operations to efficiently manage and present Morse code data.

MorseCodeApp class

The MorseCodeApp is a Tkinter-based graphical application designed to convert text to Morse code and vice versa. It initializes with dictionaries for character-to-Morse and Morse-to-character mappings. The interface includes radio buttons to select the conversion mode, text areas for input and output, and a "Generate" button to perform the conversion. The text_to_morse method converts each character to Morse code, while the morse_to_text method translates Morse code back to text, handling word and line separations. The app ensures user-friendly interaction with real-time conversion results displayed in a separate text area along with visual display of the morse code using flashlight changing from black to yellow and beeping sounds to display.

EngimaMachine class

The `EnigmaMachine` class, inherited from the `converter` class, effectively demonstrates key OOP principles such as inheritance, encapsulation, and polymorphism. By inheriting from `converter`, it can reuse and extend its methods. Encapsulation is used to keep the rotor and reflector details private, ensuring data integrity. Polymorphism allows it to enhance the `run` method from the parent class. This class simulates WWII-era Enigma encryption and decryption using rotors and a reflector, with features like user-defined rotor settings, automatic rotor rotation, text processing, and user prompts for rotor positions.

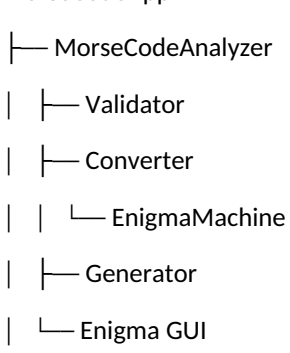
Validator class

The `validator` class in this script provides essential functions to ensure smooth program execution and prevent crashes. It initializes with private attributes for input and output files and includes methods to prompt the user to press Enter to continue and to validate file contents. The `only_has_alphabets` method checks if a file contains only alphabetic characters and spaces, while `has_no_alphabets` ensures a file does not contain any alphabetic characters, making it suitable for Morse code. The `check_alphabet_input_file_exists` and `check_morse_input_file_exists` methods prompt the user to input a valid file name, verifying its existence and content type. Lastly, `check_output_file_type` ensures the output file has a `.txt` extension, maintaining compatibility and preventing errors in file handling.

Difficulties Faced and Achievements

To me, it was difficult to initially do the encoding as my code initially I remove the last letter for each line because if the last line had a full stop, then I wanted to remove, in the end I split by word then convert by letter in word and joined them back together which work! Then for the graph, initially I wanted to use NumPy array and transpose it to get the graph from horizontal to vertical which did not work, so after much thinking I found out that using string and then rearranging the string is much easier using a loop. The GUI was tough, but I was able to maintaining smooth transitions between text and Morse code with visual and auditory feedback.

Class Diagram



Summary of Data Structures Used

Class	Data Structure	Purpose	Big O Notation	Explanation
MorseCodeAnalyzer	List	Store graph lines and Morse keywords	O(n)	Linear time complexity for iterating through lists
MorseCodeAnalyzer	Dictionary	Count word frequencies	O(1)	Constant-time complexity for dictionary lookups
MorseCodeAnalyzer	String	Manipulate and format output	O(n)	Linear time complexity for string manipulations
Conversion	Dictionary	Quick text-to-Morse lookups	O(1)	Constant-time dictionary lookup
Conversion	List, String	Store intermediate results	O(n)	Linear time complexity for list and string operations
Generator	Dictionary	Map Morse code to text	O(1)	Constant-time dictionary lookup
Generator	List, String, Set	Read and write data	O(n)	Linear time complexity for list and string operations; set operations are generally O(1)
Generator	List, Dictionary	Sort and create graphs	O(n log n)	Log-linear time complexity for sorting operations
MorseCodeApp	Dictionary	Map characters to Morse code	O(1)	Constant-time dictionary lookup
MorseCodeApp	String	Display text in UI	O(n)	Linear time complexity for string manipulations
MorseCodeApp	Tkinter Widgets	Create UI components	O(1)	Constant-time complexity for creating UI components
Validator	String	Validate contents	O(n)	Linear time complexity for string operations
Validator	File I/O	Validate files	O(n)	Linear time complexity for file reading and writing
EnigmaMachine	List	Manage rotor positions	O(n)	Linear time complexity for list operations
EnigmaMachine	String	Process text for encryption	O(n)	Linear time complexity for string manipulations

Appendix

References:

pyAudio documentation: https://people.csail.mit.edu/hubert/pyaudio/docs/

python GUI tkinter: https://www.geeksforgeeks.org/python-gui-tkinter/

How enigma machine works: https://en.wikipedia.org/wiki/Enigma_machine#Operation

Python Source Code:

main.py

```
from activation import MorseCodeAnalyzer

analyzer = MorseCodeAnalyzer()
analyzer.main()
```

activation.py

```
#############################################################################
# Importing Python Classes
#############################################################################

from validation import validator
from convertion import converter
from generation import generator
from gui_converter import MorseCodeApp
from engima import EnigmaMachine

#############################################################################
# Main Class
#############################################################################
class MorseCodeAnalyzer:

    # Initialising files
    def __init__(self):
        self.validator = validator()
        self.converter = converter()
        self.generator = generator()

    ############################################################################
    #  START PROGRAM FUNCTION
    ############################################################################

    def main(self):
        print('')
        print('*' * 50)
        print('* ST1507 DSAA: MorseCode Message Analyzer', ' ' * 6, '*')
        print('*_____________________________________________*')
        print('*                                             *')
        print('*   - Done By: Shaun Kwo Rui Yu(2317933)      *')
        print('*   - Class DAAA/2A/03                        *')
        print('*' * 50, '\n\n')
```

```
        self.validator.pressEnter()

        while True:
            choice = input('''
Please select your choice ('1','2','3','4','5','6','7'):
    1. Convert Text To Morse Code
    2. Convert Morse Code To Text
    3. Generate Morse Word Frequencies Report
    4. Generate Morse Keyword Frequencies Graph
    5. Morse Code Converter and Flashlight
    6. WWII Engima Machine
    7. Exit
Enter choice: ''')

            if choice == '1':
                self.validator.create_input_file()
                input_file = self.validator.check_alphabet_input_file_exists()
                output_file = self.validator.check_output_file_type()
                input_file, output_file =self.converter.encode_morse(input_file, output_file)
                print()
                self.validator.pressEnter()
                self.converter.display_text_files(input_file, output_file)
                self.validator.pressEnter()
                pass


            elif choice == '2':
                self.validator.create_input_file()
                input_file = self.validator.check_morse_input_file_exists()
                output_file = self.validator.check_output_file_type()
                input_file, output_file =self.converter.decode_morse(input_file, output_file)
                print()
                self.validator.pressEnter()
                self.converter.display_text_files(input_file, output_file)
                self.validator.pressEnter()
                pass


            elif choice == '3':
                input_file = self.validator.check_morse_input_file_exists()
                output_file = self.validator.check_output_file_type()
                print('\n>>>Report generation completed!\n')
                self.validator.pressEnter()
                print()
                self.converter.decode_morse(input_file, output_file)
                self.generator.generate_report(input_file, output_file)
                pass
```

```
            elif choice == '4':
                input_file = self.validator.check_morse_input_file_exists()
                output_file = self.validator.check_output_file_type()

                print('\n>>>Graph generation completed!\n')
                self.validator.pressEnter()
                print()
                self.converter.decode_morse(input_file, output_file)
                self.generator.generate_graph(input_file, output_file)

                pass
            elif choice == '5':
                gui_app = MorseCodeApp()
                gui_app.run()

            elif choice == '6':
                input_file = self.validator.check_alphabet_input_file_exists()
                output_file = self.validator.check_output_file_type()
                engima= EnigmaMachine()
                engima.run(input_file, output_file)

                pass
            elif choice == '7':
                print("Bye, thanks for using ST1507 DSAA: MorseCode Message Analyzer")
                break
            else:
                print("Invalid choice. Please enter a number from 1 to 7.")


if __name__ == "__main__":
    # This block will run only if this file is executed directly,
    # not when it's imported as a module
    analyzer = MorseCodeAnalyzer()
    analyzer.main()
```

validation.py

```
###############################################################################
# Importing Python Libraries
###############################################################################
import getpass, os

###############################################################################
#  SMALL FUNCTIONS FOR CHECKING AND STREAMLINING OF PROGRAM TO PREVENT CRASHES
###############################################################################
class validator:

    # Initialising files
```

```
    def __init__(self):
        self.__input_file=''
        self.__output_file=''
        pass

    def pressEnter(self):
        user_input = getpass.getpass("Press Enter, to continue....")
        if not user_input:
            pass
        else:
            print("You must press Enter to continue.")
            self.pressEnter()

    def only_has_alphabets(self):
        with open(self.__input_file, 'r') as file:
            input_text = file.read()
        for c in input_text:
            if not (c.isalpha() or c.isspace()):
                return False
        return True


    def check_alphabet_input_file_exists(self):
        while True:
            self.__input_file = input("\nPlease enter input file: ")
            if os.path.isfile(self.__input_file) and self.__input_file.lower().endswith('.txt') and
self.only_has_alphabets():
                return self.__input_file
            else:
                print("Invalid input file or .txt file extension or does not only have alphabets.")

    def has_no_alphabets(self):
        with open(self.__input_file, 'r') as file:
            input_text = file.read()
        for c in input_text:
            if c.isalpha():
                return False
        return True

    def check_morse_input_file_exists(self):
        while True:
            self.__input_file = input("\nPlease enter input file: ")
            if os.path.isfile(self.__input_file) and self.__input_file.lower().endswith('.txt') and
self.has_no_alphabets():
                return self.__input_file
            else:
                print("Invalid input file. Please provide a valid .txt file or morse code file\nFile
should not have alphabets.")
```

```
    def check_output_file_type(self):
        while True:
            self.__output_file = input("Please enter output file: ")
            if self.__output_file.lower().endswith('.txt'):
                return self.__output_file
            else:
                print("Output file must have a .txt extension.")

def create_input_file(self):
    file_name = input("\nIf you want to create file (no need .txt ending), please enter the file name else just press enter: ").strip()
    if not file_name:
        return None

    if not file_name.endswith(".txt"):
        file_name += ".txt"

    print("\nPlease enter the text below and press Enter to Exit. End with an empty line:")
    lines = []
    while True:
        line = input()
        if line == "":
            break
        lines.append(line)
    user_text = "\n".join(lines)

    with open(file_name, 'w') as file:
        file.write(user_text)

    print(f"{file_name} has been successfully created!")
```

convertion.py

```
################################################################################
# Importing Class
################################################################################
from validation import validator

################################################################################
#  CHOICE 1: COMPLEX MORSE ENCODER WITH TEXT WITH SENTENCES AND PARAGRAPHS
################################################################################
class converter:

    # Initialising files
    def __init__(self):
        self.__input_file = None
        self.__output_file = None
        self.validator=validator()
```

```
        # Initialising dictionaries
        self.text_morse_dict = {
            'A': '.-', 'B': '-...', 'C': '-.-.', 'D': '-..', 'E': '.', 'F': '..-.',
            'G': '--.', 'H': '....', 'I': '..', 'J': '.---', 'K': '-.-', 'L': '.-..',
            'M': '--', 'N': '-.', 'O': '---', 'P': '.--.', 'Q': '--.-', 'R': '.-.',
            'S': '...', 'T': '-', 'U': '..-', 'V': '...-', 'W': '.--', 'X': '-..-',
            'Y': '-.--', 'Z': '--..', " ": " ",
            '0': '-----', '1': '.----', '2': '..---', '3': '...--', '4': '....-',
            '5': '.....', '6': '-....', '7': '--...', '8': '---..', '9': '----.'
        }

        self.morse_text_dict={value: key for key, value in self.text_morse_dict.items()}



    def encode_morse(self, input_file, output_file):
        self.__input_file = input_file
        self.__output_file = output_file

        with open(self.__input_file, 'r') as input_file:
            input_text = input_file.read()

            encoded_line = []
        paragraphs = input_text.split('\n')

        for paragraph in paragraphs:
            words = paragraph.split()
            encoded_word = []

            for word in words:
                encoded_letter = []
                for letter in word.upper():
                    if letter in self.text_morse_dict:
                        encoded_letter.append(self.text_morse_dict[letter])

                encoded_word.append(','.join(encoded_letter))

            encoded_line.append(',  ,'.join(encoded_word))

        with open(self.__output_file, 'w') as output_file:
            output_file.write('\n'.join(encoded_line))

        return self.__input_file, self.__output_file


################################################################################
#   CHOICE 2: COMPLEX MORSE DECODER WITH TEXT WITH SENTENCES AND PARAGRAPHS
################################################################################
```

```
    def decode_morse(self, input_file, output_file):
        self.__input_file = input_file
        self.__output_file = output_file
        with open(self.__input_file, 'r') as input_file:
            encoded_text = input_file.readlines()

        decoded_text = []
        for line in encoded_text:
            decoded_line = []
            for code in line.strip().split(','):
                if code in self.morse_text_dict:
                    decoded_line.append(self.morse_text_dict[code])
                else:
                    print(f"Error: Morse code '{code}' not recognized. Please enter a file with valid Morse codes.")
                    self.__input_file = self.validator.check_morse_input_file_exists()
                    return self.decode_morse(input_file, output_file)
            decoded_text.append(''.join(decoded_line))

        with open(self.__output_file, 'w') as output_file:
            output_file.write('\n'.join(decoded_text))

        return self.__input_file, self.__output_file

    def display_text_files(self, input_file, output_file):
        self.__input_file = input_file
        self.__output_file = output_file
        with open(input_file, 'r') as input_file:
            input_content = input_file.read()

        with open(output_file, 'r') as output_file:
            content = output_file.read()

        print()
        print(f"Input File: {input_file.name}")
        print('='*100)
        print(input_content)
        print('='*100)
        print('\n')
        print(f"Output File: {output_file.name}")
        print('='*100)
        print(content)
        print('='*100)
        print()

if __name__ == '__main__':
    converter = converter()
    converter.display_text_files()
```

generation.py

```
###############################################################################
# Importing Python Libraries
###############################################################################
import numpy as np
import datetime

from validation import validator
from convertion import converter

###############################################################################
# FUNCTION TO CALCULATE AND STORE THE VARIABLES IN CHOICE 3 & 4 ARE THE SAME
###############################################################################
class generator:

    def __init__(self):
        self.__input_file = None
        self.__output_file = None
        self.validator=validator()
        self.converter=converter()

###############################################################################
#  SIMPLE ENCODER FOR CHOICE 3 AND 4
###############################################################################
    def charToMorse(self, char):
        text_morse_dict = {
            'A': '.-', 'B': '-...', 'C': '-.-.', 'D': '-..', 'E': '.', 'F': '..-.',
            'G': '--.', 'H': '....', 'I': '..', 'J': '.---', 'K': '-.-', 'L': '.-..',
            'M': '--', 'N': '-.', 'O': '---', 'P': '.--.', 'Q': '--.-', 'R': '.-.',
            'S': '...', 'T': '-', 'U': '..-', 'V': '...-', 'W': '.--', 'X': '-..-',
            'Y': '-.--', 'Z': '--..', " ": " ",
            '0': '-----', '1': '.----', '2': '..---', '3': '...--', '4': '....-',
            '5': '.....', '6': '-....', '7': '--...', '8': '---..', '9': '----.'
        }

        char = char.upper()

        # Check if letter is in keys
        if char in text_morse_dict:
            return text_morse_dict[char]

        else:
            self.validator.check_morse_input_file_exists()
            self.validator.check_output_file_type()

            print('\n>>>Report generation completed!\n')
```

```
            self.validator.pressEnter()

            self.converter.decode_morse(self.__input_file, self.__output_file)
            self.generate_report(self.__input_file, self.__output_file)


    def textToMorse(self, line):
        output = ''
        for char in line[:-1]:  # Everything except the last
            output += self.charToMorse(char) + ','
        # Do the last separately
        output += self.charToMorse(line[-1])  # Only the last one
        return output



    def store_output_data(self, input_file, output_file):
        self.__input_file = input_file
        self.__output_file = output_file
        with open(self.__output_file, 'r') as output_file:
            content = output_file.read().upper()

        with open('./stopwords.txt', 'r') as stopword_file:
            stopwords = set(stopword_file.read().upper().split())

        # Find unique words in output content
        words = set(content.split())

        # Initialise dictionary to store KEY: VALUE which is WORD: COUNT
        word_frequency = {}
        for word in words:
            if word.isalpha():
                # Do a count for each word
                word_frequency[word] = content.split().count(word)


        # Sort words by frequency in ascending order
        sorted_by_frequency = sorted(word_frequency.items(), key=lambda x: x[1])

        # Organize words by frequency for printing
        words_by_frequency = {}
        for word, frequency in sorted_by_frequency:
            if frequency not in words_by_frequency:
                words_by_frequency[frequency] = []
            words_by_frequency[frequency].append((word, self.textToMorse(word)))

        # Append keywords not in stopwords to the keywords list
        keywords = [word for word in word_frequency if word not in stopwords]
```

```
        # Sort keywords by frequency in descending order
        keywords.sort(key=lambda x: word_frequency[x], reverse=True)

        return content, words_by_frequency, sorted_by_frequency, stopwords, keywords, word_frequency


################################################################################
#   CHOICE 3
################################################################################
    def generate_report(self, input_file, output_file):
        content, words_by_frequency, sorted_by_frequency, stopwords, keywords, word_frequency =
self.store_output_data(input_file, output_file)

        current_time = datetime.datetime.now()
        report = f"                                REPORT GENERATED ON: {current_time.strftime('%d%m-%Y
%H:%M')}"

        report_header = [
            "*" * 100,
            report,
            "*" * 100,
            "",
            '*** Decoded Morse Text',
            content,
            '\n'
        ]

        report_body = []
        for frequency, word_list in words_by_frequency.items():
            report_body.append(f'*** Morse Words with frequency=> {frequency}')
            sorted_by_morse_length = sorted(word_list, key=lambda x: (-len(x[1]), x[0]))
            for word, morse in sorted_by_morse_length:
                label = '(*)' if word not in stopwords else ''
                report_body.append(f'[{morse}]=> {word} {label}')
            report_body.append('')

        report_body.append('*** Keywords sorted by frequency')
        for keyword in keywords:
            frequency = word_frequency[keyword]
            report_body.append(f'{keyword}({frequency})')

        full_report = report_header + report_body

        with open(self.__output_file, 'w') as output_file:
            output_file.write('\n'.join(full_report))

        with open(self.__output_file, 'r') as output_file:
            display_report = output_file.read()
```

```
            print(display_report)

##############################################################################
# CHOICE 4
##############################################################################
    def generate_graph(self, input_file, output_file):
        content, words_by_frequency, sorted_by_frequency, stopwords, keywords, word_frequency =
self.store_output_data(input_file, output_file)

        current_time = datetime.datetime.now()
        graph = f"                                GRAPH GENERATED ON: {current_time.strftime('%d%m-%Y
%H:%M')}"
        graph_header = f"""
{'*' * 100}
{graph}
{'*' * 100}

"""



        graph_body = ""
        max_freq = max(word_frequency.values())
        max_spacing = ' ' * max_freq

        for idx, keyword in enumerate(keywords):
            stars = '*' * word_frequency[keyword]
            spaces = ' ' * (max_freq - word_frequency[keyword])
            morse_code = self.textToMorse(keyword)

            graph_body+=f'{spaces}{stars}-{morse_code}\n'
            graph_body+=f'{max_spacing}-{keyword}\n'
            graph_body+=f'{max_spacing}-\n' * 8

        lines = graph_body.split('\n')
        max_len = max(len(line) for line in lines)

        # Create a 2D list with each line padded to the maximum length
        padded_lines = [line.ljust(max_len) for line in lines]

        transposed_body_graph = ''
        for i in range(max_len):
            for line in padded_lines:
                if i < len(line):
                    transposed_body_graph += line[i]
                else:
                    transposed_body_graph += '  '
            transposed_body_graph += '\n'
```

```
        full_graph = graph_header + transposed_body_graph

        with open(self.__output_file, 'w') as output_file:
            output_file.write(full_graph)


        with open(self.__output_file, 'r') as output_file:
            display_graph = output_file.read()
            print(display_graph)


if __name__ == '__main__':
    generator = generator()
    generator.generate_graph('morse.txt', 'graph.txt')
```

gui_converter.py

```
################################################################################
#   Importing Python Libraries
################################################################################
import time, threading, pyaudio
import tkinter as tk
import numpy as np
################################################################################
#   Choice 5: Morse Code GUI
################################################################################
class MorseCodeApp:

    def __init__(self):
        self.root = tk.Tk()
        self.playing=False
        self.root.title("Morse Code Converter and Flashlight")
        self.root.geometry("800x800")
        self.root.config(bg="#f0f0f0")

        self.text_morse_dict = {
            'A': '.-', 'B': '-...', 'C': '-.-.', 'D': '-..', 'E': '.', 'F': '..-.', 'G': '--.', 'H':
'....',
            'I': '..', 'J': '.---', 'K': '-.-', 'L': '.-..', 'M': '--', 'N': '-.', 'O': '---', 'P': '.
--.',
            'Q': '--.-', 'R': '.-.', 'S': '...', 'T': '-', 'U': '..-', 'V': '...-', 'W': '.--', 'X': '
-..-',
            'Y': '-.--', 'Z': '--..', '1': '.----', '2': '..---', '3': '...--', '4': '....-', '5':
'.....',
            '6': '-....', '7': '--...', '8': '---..', '9': '----.', '0': '-----',' ': ' '
        }

        self.morse_text_dict = {value: key for key, value in self.text_morse_dict.items()}
```

```
        self.playing = False  # Flag to control playback
        self.create_widgets()

    def text_to_morse(self, input_text):
        encoded_line = []
        paragraphs = input_text.split('\n')

        for paragraph in paragraphs:
            words = paragraph.split()
            encoded_word = []

            for word in words:
                encoded_letter = []
                for letter in word.upper():
                    if letter in self.text_morse_dict:
                        encoded_letter.append(self.text_morse_dict[letter])
                    else:
                        tk.messagebox.showerror('Error', f"Character '{letter}' cannot be converted to Morse code.")
                        return ""

                encoded_word.append(','.join(encoded_letter))

            encoded_line.append('  '.join(encoded_word))

        return '\n'.join(encoded_line)

    def morse_to_text(self, input_text):
        decoded_text = []
        for line in input_text.split('\n'):
            decoded_line = []
            for code in line.split(','):
                if code in self.morse_text_dict:
                    decoded_line.append(self.morse_text_dict[code])
                else:
                    tk.messagebox.showerror('Error', f"Morse code '{code}' not recognized. Please enter alphabetic Morse codes.")
                    return ''
            decoded_text.append(''.join(decoded_line))

        return '\n'.join(decoded_text)

    def convert_text(self):
        self.stop_playing()  # Stop any ongoing playback
        mode = self.mode_var.get()
        input_text = self.input_text.get("1.0", tk.END).strip()
        output_text = ""
```

```
        if mode == "text_to_morse":
            output_text = self.text_to_morse(input_text)
        else:
            output_text = self.morse_to_text(input_text)

        self.output_text.config(state=tk.NORMAL)
        self.output_text.delete("1.0", tk.END)
        self.output_text.insert(tk.END, output_text)
        self.output_text.config(state=tk.DISABLED)

        # Flash Morse code if conversion is successful
        if mode == "text_to_morse" and output_text:
            self.start_flashing(output_text)
        elif mode == "morse_to_text" and input_text and output_text:
            self.start_flashing(input_text)

    def play_tone(self, frequency, duration):
        p = pyaudio.PyAudio()
        volume = 0.5
        fs = 44100

        samples = (np.sin(2 * np.pi * np.arange(fs * duration) * frequency / fs)).astype(np.float32)

        stream = p.open(format=pyaudio.paFloat32,
                        channels=1,
                        rate=fs,
                        output=True)

        stream.write(volume * samples)

        # Close the stream after playing
        stream.stop_stream()
        stream.close()


    def flash_dot(self, label):
        label.config(bg="yellow")
        label.update()
        threading.Thread(target=self.play_tone, args=(1000, 0.2)).start()
        time.sleep(0.2)
        label.config(bg="black")
        label.update()
        time.sleep(0.2)

    def flash_dash(self, label):
        label.config(bg="yellow")
        label.update()
        threading.Thread(target=self.play_tone, args=(1000, 0.6)).start()
```

```
        time.sleep(0.6)
        label.config(bg="black")
        label.update()
        time.sleep(0.2)

    def flash_space(self, label):
        time.sleep(0.8)

    def morse_flash(self, morse_code, label):
        self.playing = True
        for char in morse_code:
            if not self.playing:
                break
            if char == '.':
                self.flash_dot(label)
                time.sleep(0.2)
            elif char == '-':
                self.flash_dash(label)
                time.sleep(0.6)
            elif char == ' ':
                self.flash_space(label)
                time.sleep(0.8)
            elif char == ',':
                time.sleep(0.2)
        self.playing = False

    def start_flashing(self, morse_code):
        self.playing = True
        threading.Thread(target=self.morse_flash, args=(morse_code, self.flashlight_label)).start()

    def stop_playing(self):
        self.playing = False

    def clear_textboxes(self, *args):
        self.input_text.delete("1.0", tk.END)
        self.output_text.config(state=tk.NORMAL)
        self.output_text.delete("1.0", tk.END)
        self.output_text.config(state=tk.DISABLED)
        self.stop_playing()

    def exit_program(self):
        self.stop_playing()
        self.root.quit()  # Stop the main event loop
        self.root.destroy()  # Destroy the window


    def create_widgets(self):
        # Create and configure the title label
```

```
        title_label = tk.Label(self.root, text="Morse Code Converter and Flashlight",
font=("Helvetica", 18, "bold"), bg="#f0f0f0")
        title_label.pack(pady=10)

        # Create a frame to hold the mode selection radio buttons
        mode_frame = tk.Frame(self.root, bg="#f0f0f0")
        mode_frame.pack(pady=10)

        # Create a StringVar to hold the mode selection and set its default value
        self.mode_var = tk.StringVar(value="text_to_morse")
        self.mode_var.trace_add("write", self.clear_textboxes)

        # Create and configure the "Text to Morse" radio button
        text_to_morse_rb = tk.Radiobutton(mode_frame, text="Text to Morse", variable=self.mode_var,
                                          value="text_to_morse", bg="#f0f0f0", font=("Helvetica", 12))
        text_to_morse_rb.grid(row=0, column=0, padx=20)

        # Create and configure the "Morse to Text" radio button
        morse_to_text_rb = tk.Radiobutton(mode_frame, text="Morse to Text", variable=self.mode_var,
                                          value="morse_to_text", bg="#f0f0f0", font=("Helvetica", 12))
        morse_to_text_rb.grid(row=0, column=1, padx=20)

        # Create a frame to hold the input and output text boxes
        text_frame = tk.Frame(self.root, bg="#f0f0f0")
        text_frame.pack(pady=20)

        # Create and configure the input label
        input_label = tk.Label(text_frame, text="Input Box", font=("Helvetica", 12), bg="#f0f0f0")
        input_label.grid(row=0, column=0, sticky="w", padx=10)

        # Create and configure the input text box
        self.input_text = tk.Text(text_frame, height=5, width=70, font=("Helvetica", 12))
        self.input_text.grid(row=1, column=0, padx=10, pady=10)

        # Create and configure the output label
        output_label = tk.Label(text_frame, text="Display Output Box", font=("Helvetica", 12),
bg="#f0f0f0")
        output_label.grid(row=2, column=0, sticky="w", padx=10)

        # Create and configure the output text box (initially disabled)
        self.output_text = tk.Text(text_frame, height=5, width=70, font=("Helvetica", 12),
state=tk.DISABLED)
        self.output_text.grid(row=3, column=0, padx=10, pady=10)

        # Create and configure the flashlight title label
        flashlight_title_label = tk.Label(self.root, text="Morse Code Flashlight", font=("Helvetica",
18, "bold"), bg="#f0f0f0")
        flashlight_title_label.pack(pady=10)
```

```
        # Create and configure the flashlight label (initially black)
        window_width = self.root.winfo_reqwidth()
        self.flashlight_label = tk.Label(self.root, bg="black", width=window_width, height=10)
        self.flashlight_label.pack(pady=20)

        # Create a frame to hold the buttons
        button_frame = tk.Frame(self.root, bg="#f0f0f0")
        button_frame.pack(pady=20)

        # Create and configure the "Generate" button
        generate_button = tk.Button(button_frame, text="Generate", command=self.convert_text,
bg="#4CAF50", fg="white",
                                    font=("Helvetica", 12), padx=20, pady=10)
        generate_button.pack(side=tk.LEFT, padx=10)

        # Create and configure the "Exit" button
        exit_button = tk.Button(button_frame, text="Exit", command=self.exit_program, bg="red",
fg="white",
                                font=("Helvetica", 12), padx=20, pady=10)
        exit_button.pack(side=tk.RIGHT, padx=10)


    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    app = MorseCodeApp()
    app.run()
```

enigma.py

```
##############################################################################
# Importing Class
##############################################################################
from convertion import converter

##############################################################################
#  CHOICE 6: WII Enigma Machine
##############################################################################

class EnigmaMachine(converter):
    def __init__(self, rotor=None):
        # Initialize the parent class
        super().__init__()
        # Initialize the rotors with lists of characters from 'A' to 'Z'
        self.__rotors = [[chr(i) for i in range(65, 91)] for _ in range(3)]
        # Set up predefined reflector mapping
```

```
        self.__reflector = {'A': 'Y', 'B': 'R', 'C': 'U', 'D': 'H', 'E': 'Q', 'F': 'S', 'G': 'L', 'H':
'D',
                            'I': 'P', 'J': 'X', 'K': 'N', 'L': 'G', 'M': 'O', 'N': 'K', 'O': 'M', 'P':
'I',
                            'Q': 'E', 'R': 'B', 'S': 'F', 'T': 'Z', 'U': 'C', 'V': 'W', 'W': 'V', 'X':
'J',
                            'Y': 'A', 'Z': 'T'}
        if rotor:
            self.set_rotor(rotor)
        else:
            self.__rotor = [0, 0, 0]

    def set_rotor(self, positions):
        self.__rotor = positions

    def rotate_rotors(self):
        # Rotate the first rotor and check for full rotation
        self.__rotor[0] = (self.__rotor[0] + 1) % 26
        if self.__rotor[0] == 0:
            # Rotate the second rotor if the first has completed a full rotation
            self.__rotor[1] = (self.__rotor[1] + 1) % 26
            if self.__rotor[1] == 0:
                # Rotate the third rotor if the second has completed a full rotation
                self.__rotor[2] = (self.__rotor[2] + 1) % 26

    def encode_decode(self, text):
        output = ''
        for letter in text:
            if letter.isalpha():
                for i in range(3):
                    pos = (ord(letter.upper()) - 65 + self.__rotor[i]) % 26
                    letter = self.__rotors[i][pos]

                # Reflect the letter
                letter = self.__reflector[letter]
                for i in range(2, -1, -1):
                    # In the Enigma machine, letters pass through the rotors from right to left during
decoding,
                    # so we start from the third rotor (index 2) and move towards the first rotor
(index 0).
                    pos = (self.__rotors[i].index(letter) - self.__rotor[i] + 26) % 26
                    letter = chr(pos + 65)
            output += letter
            self.rotate_rotors()
        return output

    def prompt_rotor(self):
        rotor = []
```

```
        for i in range(3):
            while True:
                try:
                    pos = int(input(f"Enter rotor {i+1} position (1-26): "))
                    if pos < 1 or pos > 26:
                        raise ValueError("Position must be between 1 and 26.\n")
                    rotor.append(pos - 1)  # Convert to 0-based index
                    break
                except ValueError as e:
                    print(e)
        return rotor

    def run(self, input_file, output_file):
        rotor = self.prompt_rotor()
        self.set_rotor(rotor)

        with open(input_file, 'r') as file:
            plaintext = file.read()

        print()
        ciphertext = self.encode_decode(plaintext)

        with open(output_file, 'w') as file:
            file.write(ciphertext)

        print(">>>Encryption complete. Check the output file for the ciphertext.")
        self.display_text_files(input_file, output_file)

# Save this file as enigma.py
if __name__ == "__main__":
    input_file = input("Enter the input file path: ")
    output_file = input("Enter the output file path: ")
    enigma = EnigmaMachine()
    enigma.run(input_file, output_file)
```

~ The End ~