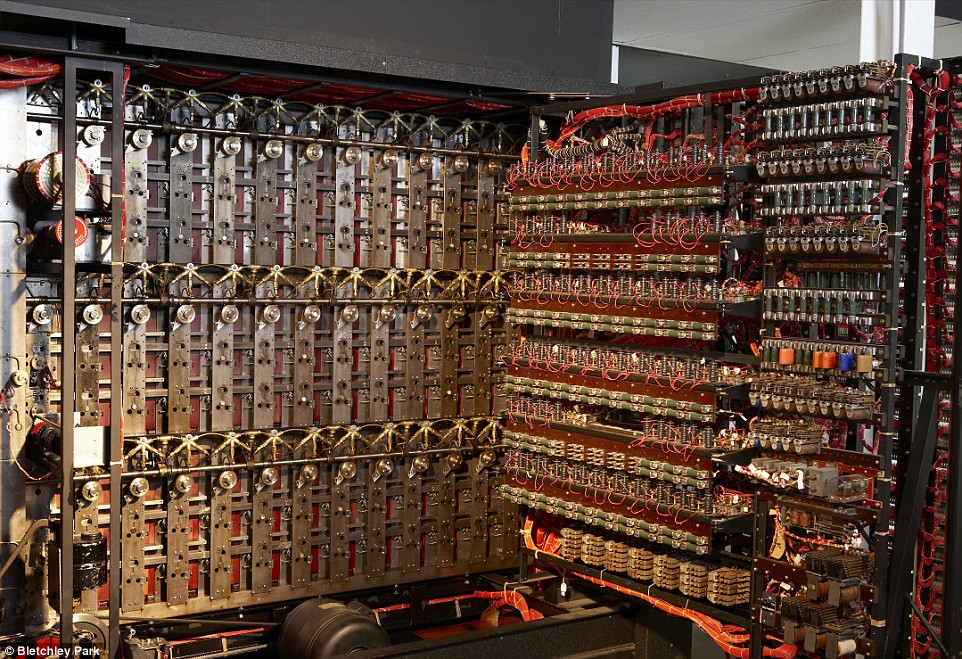
10/01/2019

EFREI PARIS

Projet de Python :

Enigma – La « Bombe »

***Enigma et la « Bombe »***



Enigma et la « bombe » sont deux formidables et complexes machines qui ont servies à coder et craquer des messages durant la seconde Guerre Mondiale. Enigma était la machine utilisée par les Allemands afin de coder un message, en utilisant les conditions initiales données le jour d’avant les Allemand pouvaient déchiffrer le code. Allan Turing, qui possédait une machine Enigma grâce au polonais qui l’avait recopiée secrètement, avait pour rôle de trouver les conditions initiales du jour. Il devait y parvenir en moins de 24 h car tous les jours ces conditions changeaient et craquer le message Allemands devenait donc impossible. Seulement, une attaque du code par brutforcing semblait impossible car trop compliquée à réaliser et trop long au niveau temps pour déchiffrer ne serait-ce qu’une journée de code.

Comment faire ? Nous allons le découvrir….

**Sommaire**

**I ENIGMA**

**La base du code / ce que nous avons dû rajouter** ………………………………………………………

**Enigma statique (code / décode)**…………………………………………………………………………………

**Enigma Dynamique (code / décode)**..........................................................................…...

**Fonction Turing**…………………………………………………………………………………………………………...

**II TKINTER : ENIGMA\_CODE**

**Création de la zone de texte** …………………………………………………………................................

**Création des boutons** …………………………………………………………...........................................

**Création des rotors – couleurs...**………………………………………..............................................

**III TKINTER : ENIGMA\_DECODE**

**Création des boutons – Fonction Turing** ……………………………...........................................

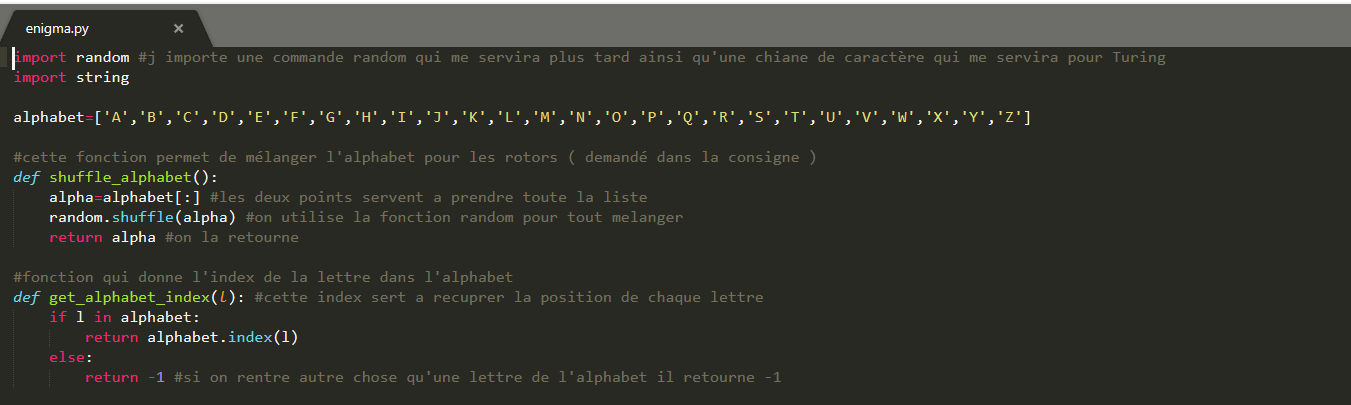
**Conditions initiales** ………………………………………………………….............................................

**IV CONCLUSION**

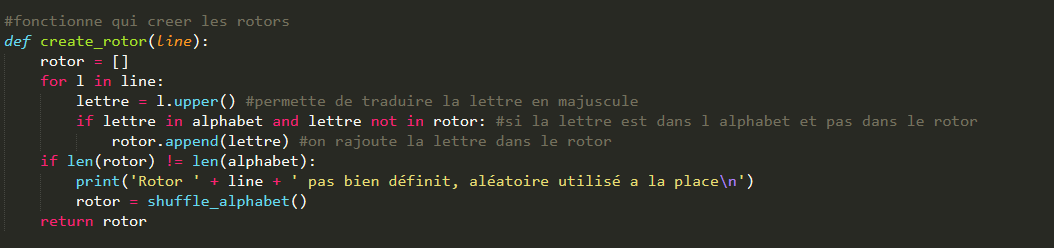


**Enigma : La base du code**

Pour commencer Enigma, avant même l’interface graphique nous avons dû créer quelques petites fonctions ou apporter des « conditions » préalables pour un bon fonctionnement de notre code. En effet il fallait commencer par importer les deux commandes « random » et la chaine de caractère « string » qui nous servirons plus tard. Ensuite, la consigne nous demandait de créer une fonction « shuffle\_alphabet » qui sert à mélanger l’alphabet grâce a la fonction random (random.shuffle). Cela nous servira pour les rotors. Nous avons aussi créé une fonction « get\_alphabet\_index » qui sert à créer et utiliser un index de notre alphabet c’est-à-dire de pouvoir récupérer la position de chaque lettre.

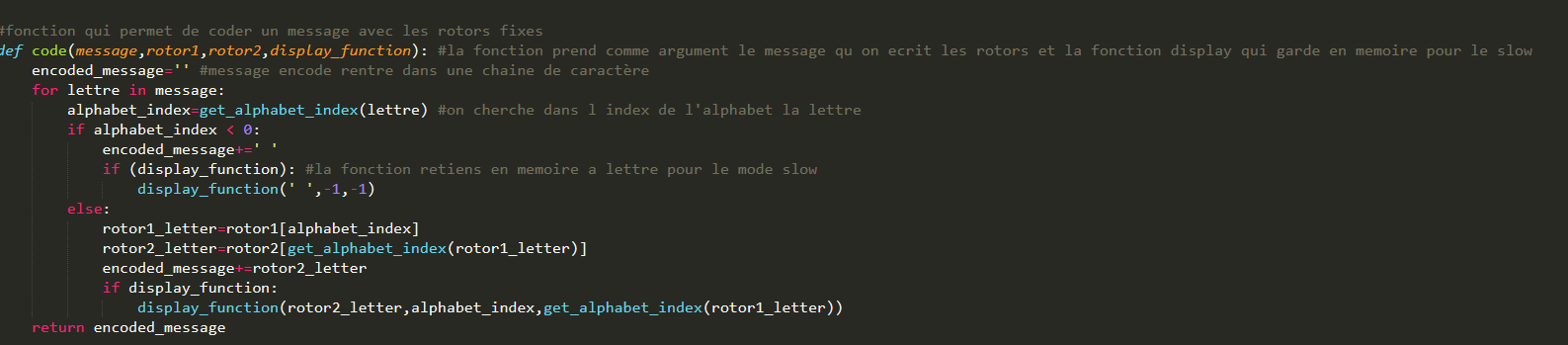


Il restait maintenant une dernière étape avant de pouvoir commencer à coder Enigma. La création de nos rotors. Le rotor va donc pour toutes les lettres de la ligne les « upper » c’est-à-dire les traduire de minuscule à majuscule. Si l’utilisateur écrit le message en minuscule grâce à la commande « L.upper » le texte sera traduit et lisible par le programme en majuscule. On a ensuite dit, avec notre boucle « if » que si la lettre fait partie de l’alphabet mais pas du rotor alors grâce à « rotor.append » qui est une commande qui sert ajouter un élément, on rajoute la lettre dans le rotor. Finalement, « rotor=shuffle\_alphabet » existe pour dire que le rotor va utiliser la fonction shuffle alphabet donc être à son tour mélangé.





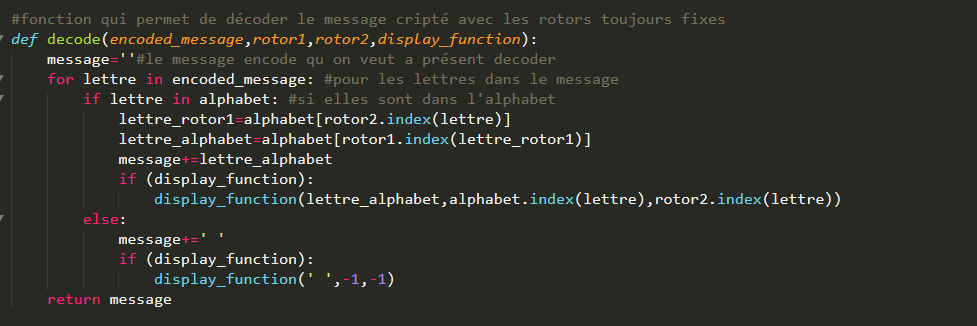
Passons maintenant au codage simple sans les rotors qui bougent. La consigne nous demandait effectivement de pouvoir coder avec des rotors qui ne bougent pas.



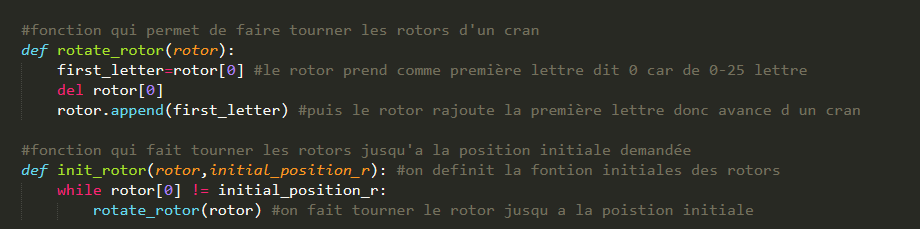
Nous créons donc la fonction code qui va avoir comme argument le message à coder les deux rotors ainsi que la « display\_function » qui va juste être utile pour le slow mode car elle se charge de garder en mémoire la lettre. Ensuite, comme on peut le voir avec la boucle « for », pour toute les lettre du message il va utiliser la commande de l’index de l’alphabet afin de trouver chaque lettre du message et e pouvoir les utiliser pour le coder. « Encode\_message » est donc une chaine de caractère symbolisée par les deux guillemets. On voit ici dans le else que les deux rotors fixes vont chercher les lettre puis encoder le message et finalement le retourner.

Passons ensuite au décodage simple sans les rotors qui bougent. La consigne nous demandait effectivement de pouvoir décoder avec des rotors qui ne bougent pas. Comme nous pouvons le remarquer ci-dessous la fonction décode et code fonctionne à peu près du même principe à l’exception près qu’elles font l’inverse puisqu’elles sont coder en miroir. On utilise les m^mes étapes dans un ordre différent.

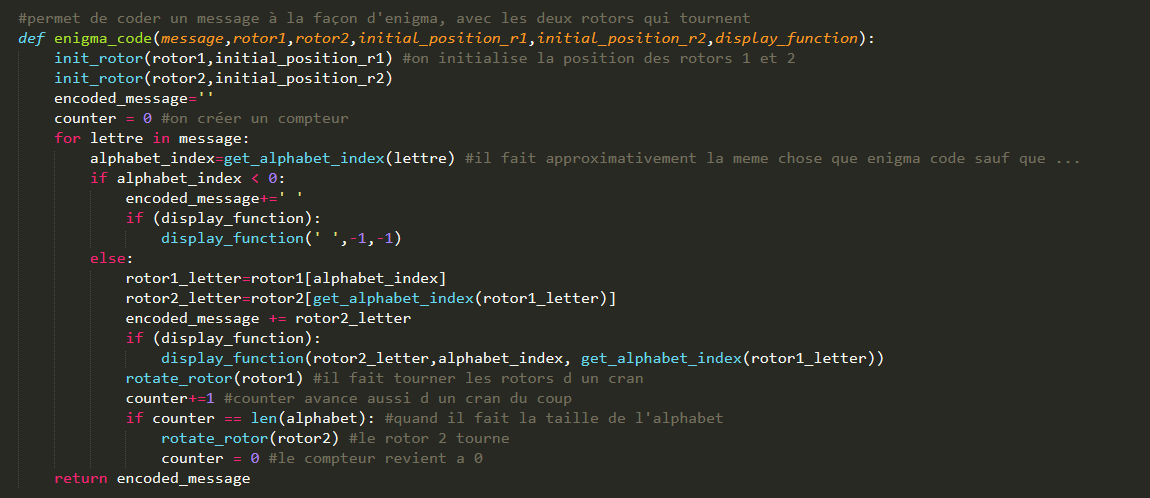
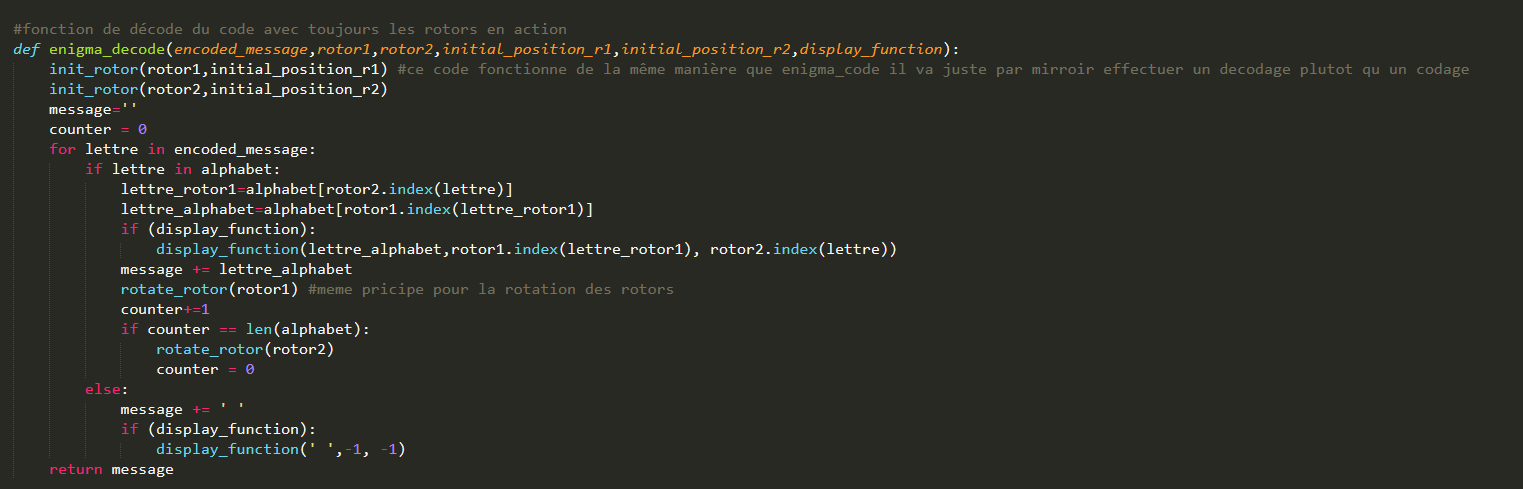




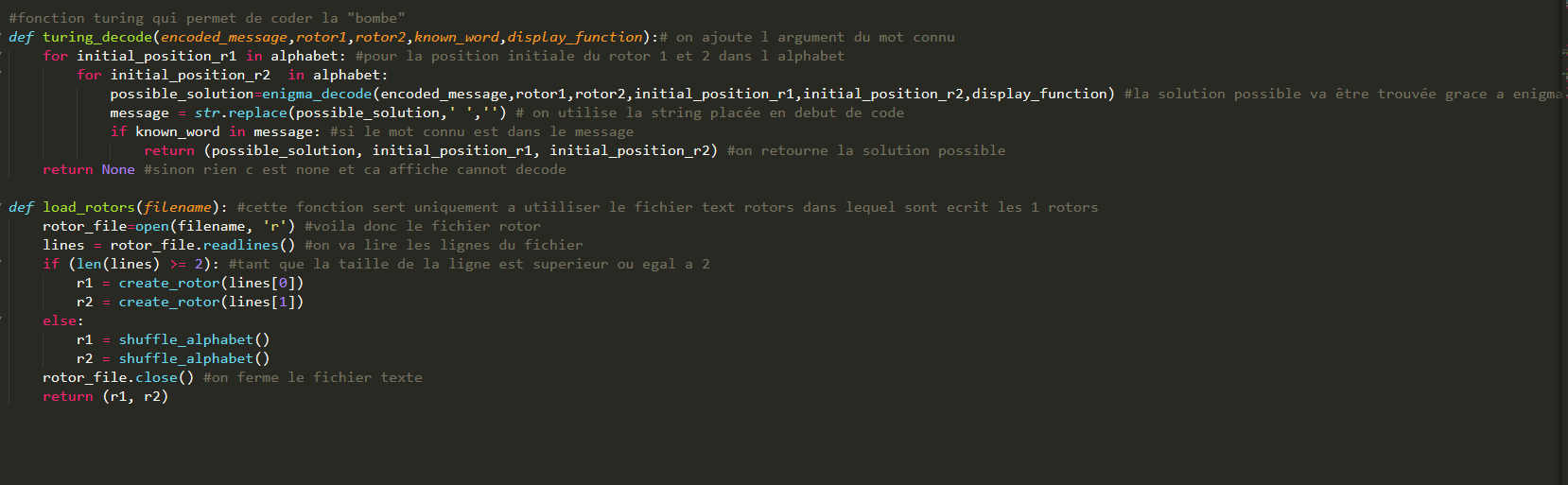
Nous devions ensuite refaire ces deux étapes mais avec des rotors qui bougent. J’ai donc créer une fonction qui permet de faire tourner les rotors :



Puis ensuite j’ai code en utilisant le code de la fonction code et décode ainsi que la fonction rotate\_rotor les systèmes pour coder et décoder avec les rotors qui bougent



Encore une fois les deux fonctionnent de la même manière mais en miroir. J’ai finalement codé la fonction turing\_decode qui à l’image de « la bombe » décode le message à l’aide d’un mot probable.



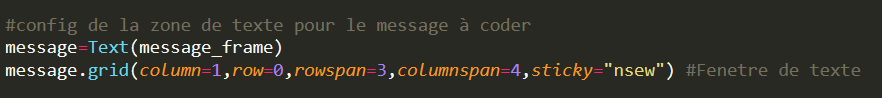


**Tkinter : Enigma\_Code**

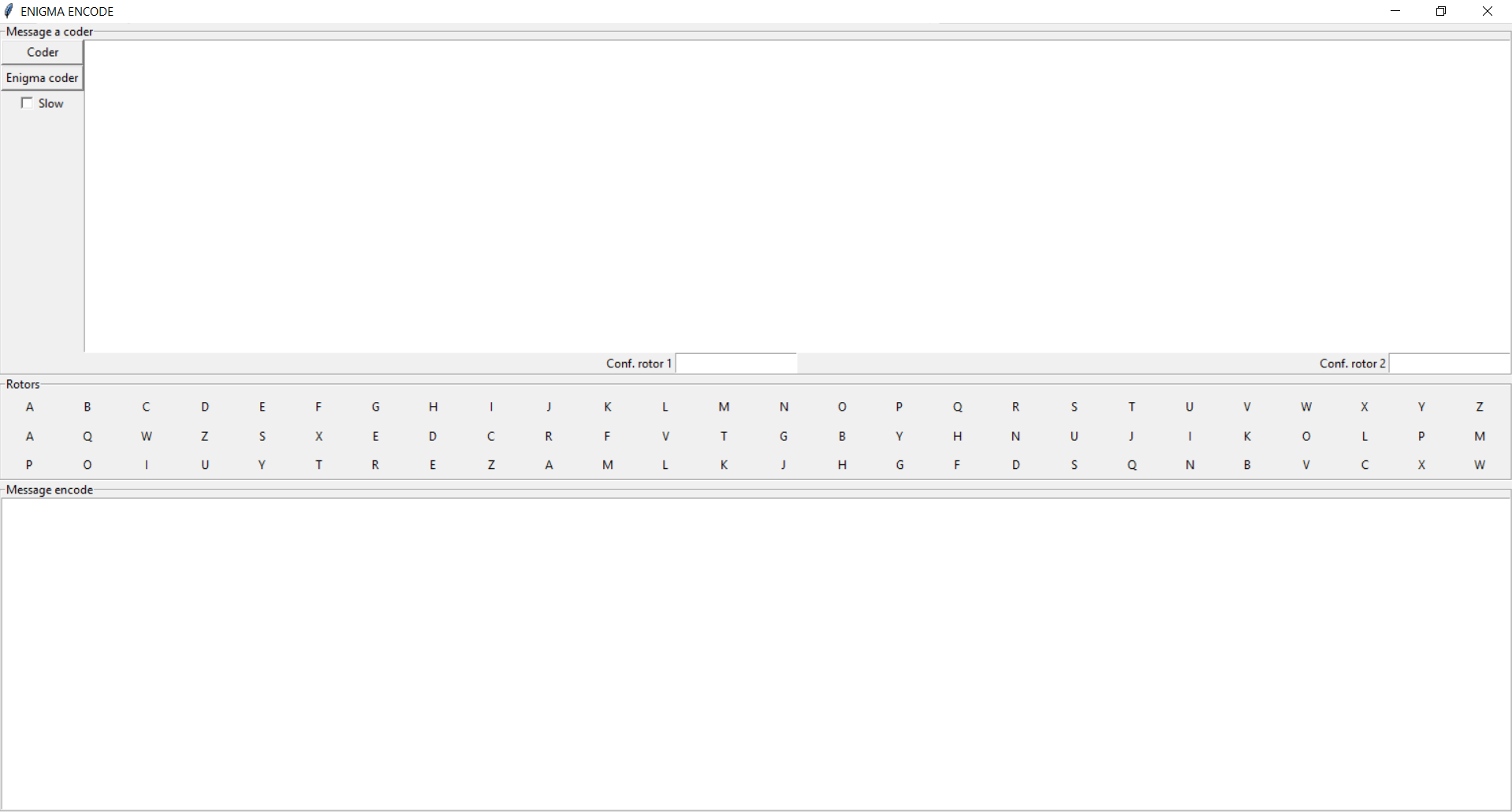
Durant cette partie, nous allons vous parlez de l’interface graphique de Enigma\_Code. Nous avons utilisé la bibliothèque Tkinter afin de créer celle-ci. N’ayant jamais utilisé cette bibliothèque, la partie graphique a été l’une des étapes les plus difficiles à appréhender et à réaliser.

**Création de la zone de texte**

Afin de pouvoir permettre à l’utilisateur de rentrer le mot ou la phrase qu’il souhaitait coder il a fallu que nous créions une « Zone de texte » à l’aide de la fonction Text () qui est comprise dans la bibliothèque Tkinter.

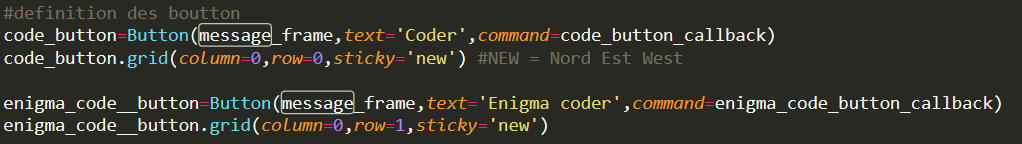


Cette fonction prend en argument la variable « message\_frame » qui sera rempli par l’utilisateur. Nous avons appelé cette zone de texte « message » et avons dû définir sa taille ainsi que sa position dans l’interface graphique.

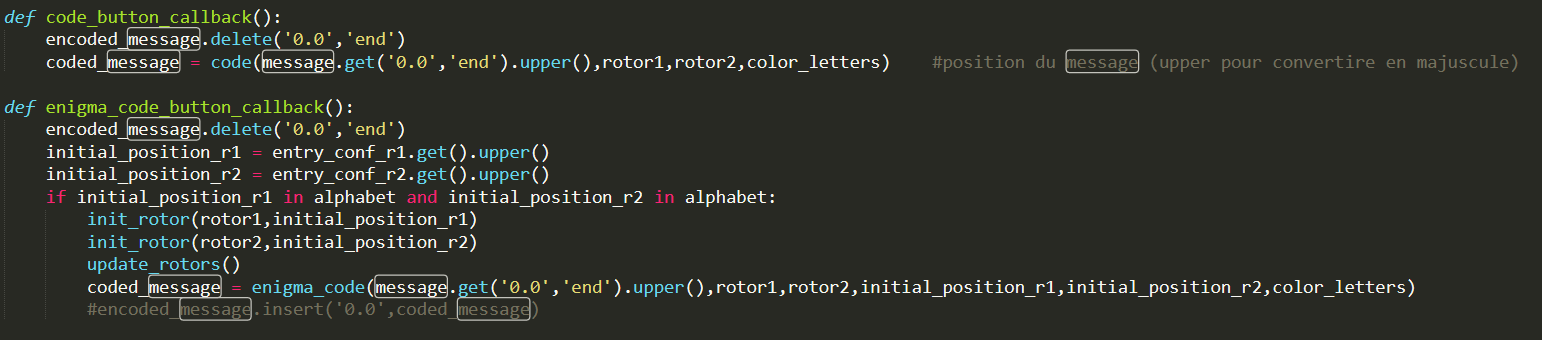


**Création des boutons**

Après que l’utilisateur a rempli la zone de texte, il dispose de 2 boutons et d’une « checkbox ». Les deux boutons sont les boutons Coder et Enigma coder. Nous avons créé ces boutons à l’aide de la fonction Button() qui est comprise elle aussi dans la bibliothèque Tkinter.

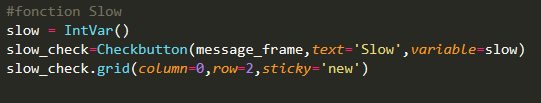


Cette fonction prend en argument la variable « message\_frame » ainsi que la fonction « code\_button\_callback »pour le bouton Coder et la fonction « enigma\_code\_button\_callback » pour le bouton Enigma coder.



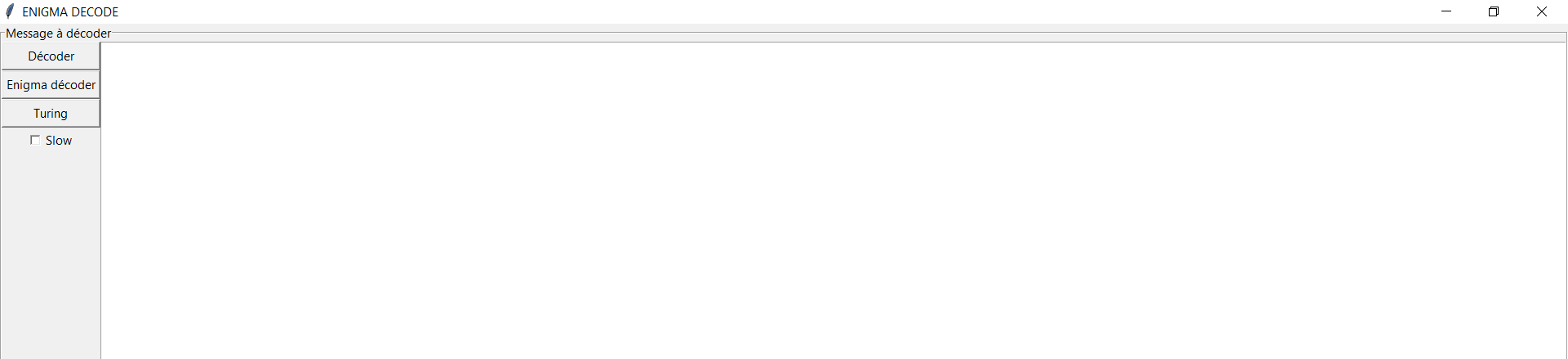
Ses fonctions ont pour but de récupérer le message rentré par l’utilisateur et d’y appliqué la fonction code pour le bouton Coder et la fonction enigma\_code pour le bouton Enigma coder. La fonction delete() comprise dans ces deux fonctions permet de supprimer puis remplacer le message codé lorsque l’utilisateur change son message et clique à nouveau sur l’un des deux boutons. De plus aillant laisser l’opportunité à l’utilisateur de choisir les positions initiales des rotors. Nous avons décidé qu’il serait indispensable de choisir ses positions afin d’utiliser le bouton Enigma coder.

On peut ensuite trouver une « checkbox » nommée Slow. Elle a été créé à l’aide de la fonction Checkbutton().



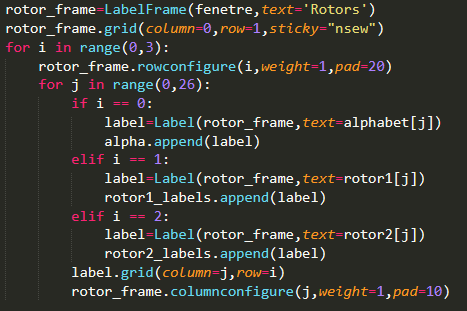


Cette fonction permet de voir étape par étape le codage du message rentré par l’utilisateur. Lorsque l’utilisateur coche cette case, un changement que nous verrons après s’opère au niveau des rotors.



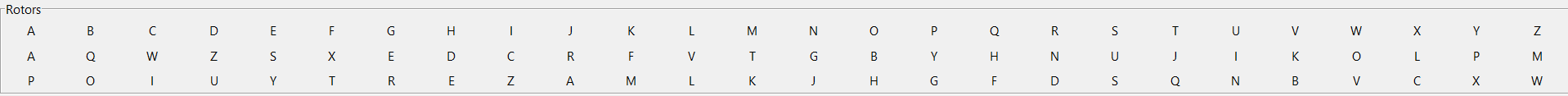
**Création des rotors**

Lorsque l’utilisateur clique sur l’un des deux boutons, les fonctions présentées en première partie s’activent et code alors le message entré par l’utilisateur. Nous avons voulu montrer comment fonctionner le code et avons pour cela représenté les rotors ainsi que l’alphabet dans l’ordre.

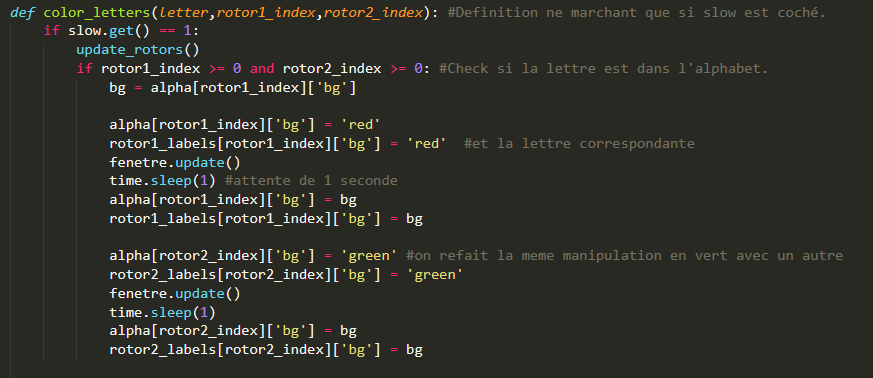


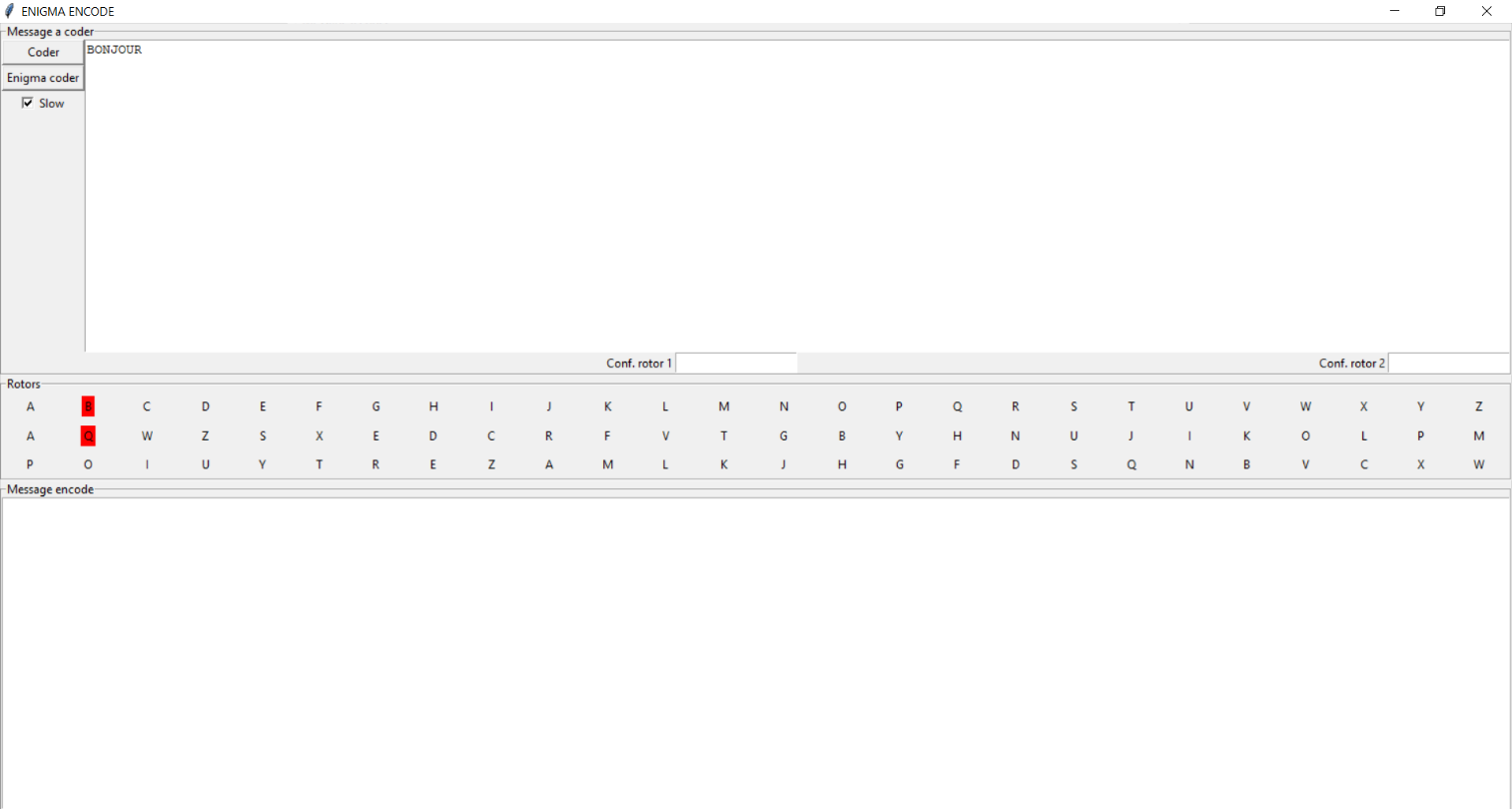
Nous avons réalisé une boucle « pour » afin d’écrire lettres par lettres les différentes listes. Puis avons écrit 3 lignes : 1 avec l’alphabet, 1 avec le rotor1 et 1 avec le rotor2.





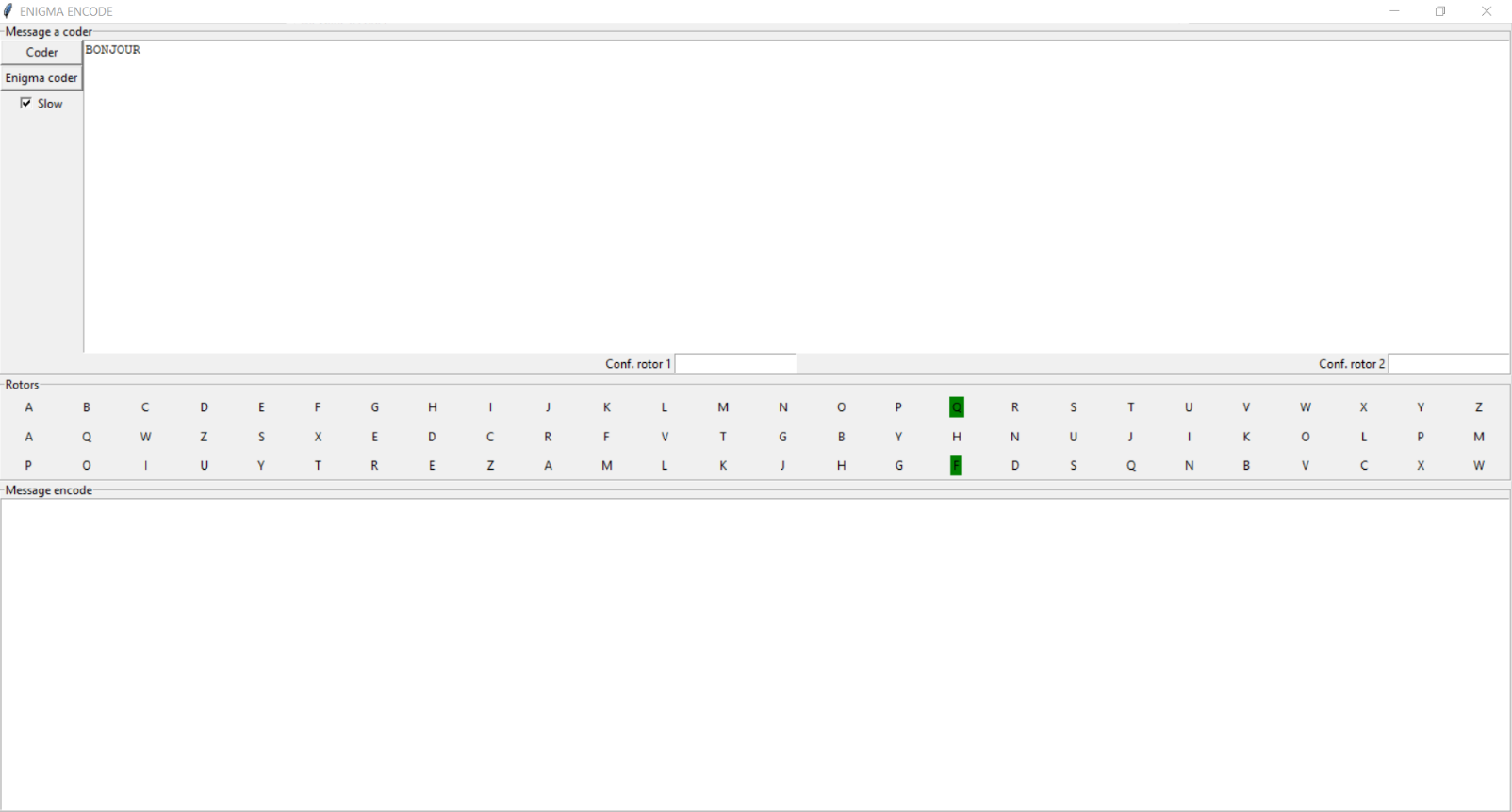
Pour finir nous voulions que l’utilisateur puisse voir comment était codé son message. Nous avons donc ajouté la fonction Slow dans cette optique et avons décidé de montrer ce qu’il se passait dans chaque étape en colorant les lettres nécessaires au codage.



Grace à Tkinter, nous pouvions juste rendre en couleur le fond des lettres grâce à l’appellation « bg » signifiant background. Nous avons utilisé alors cette appellation pour montrer la lettre entrer par l’utilisateur dans l’alphabet puis en quoi elle était transposée dans le rotor1 et enfin en quoi elle était codée avec le rotor2. Nous avons dû réaliser un .update de notre fenêtre afin qu’elle montre chaque lettre. Enfin nous avons utilisé la fonction time.sleep() qui a permis de laisser un délai d’une seconde entre chaque apparition des lettres.



On voit ici que le B est surligné en rouge avec le Q qui est placé au même index sur le rotor 1.



Puis on récupère la position du Q dans l’alphabet et on l’associe à la même position dans le rotor2, ce qui nous donne un F. On peut alors voir les lettres Q et F surligné en vert.



**Tkinter : Enigma\_Decode**

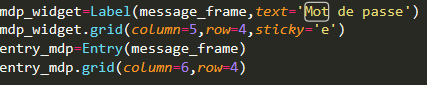
Durant cette partie, nous allons vous parlez de l’interface graphique de Enigma\_Decode. Tout comme pour l’interface graphique Enigma\_Code nous avons utilisé Tkinter pour réaliser cette interface graphique. La procédure suivie a été la même pour les zones de textes ainsi que les boutons et les rotors. Mais pour Enigma\_Decode, nous devions créer une fonction Turing() qui permettrait à l’utilisateur de décoder un message sans connaitre les conditions initiales. C’est sur cela que cette partie portera principalement.

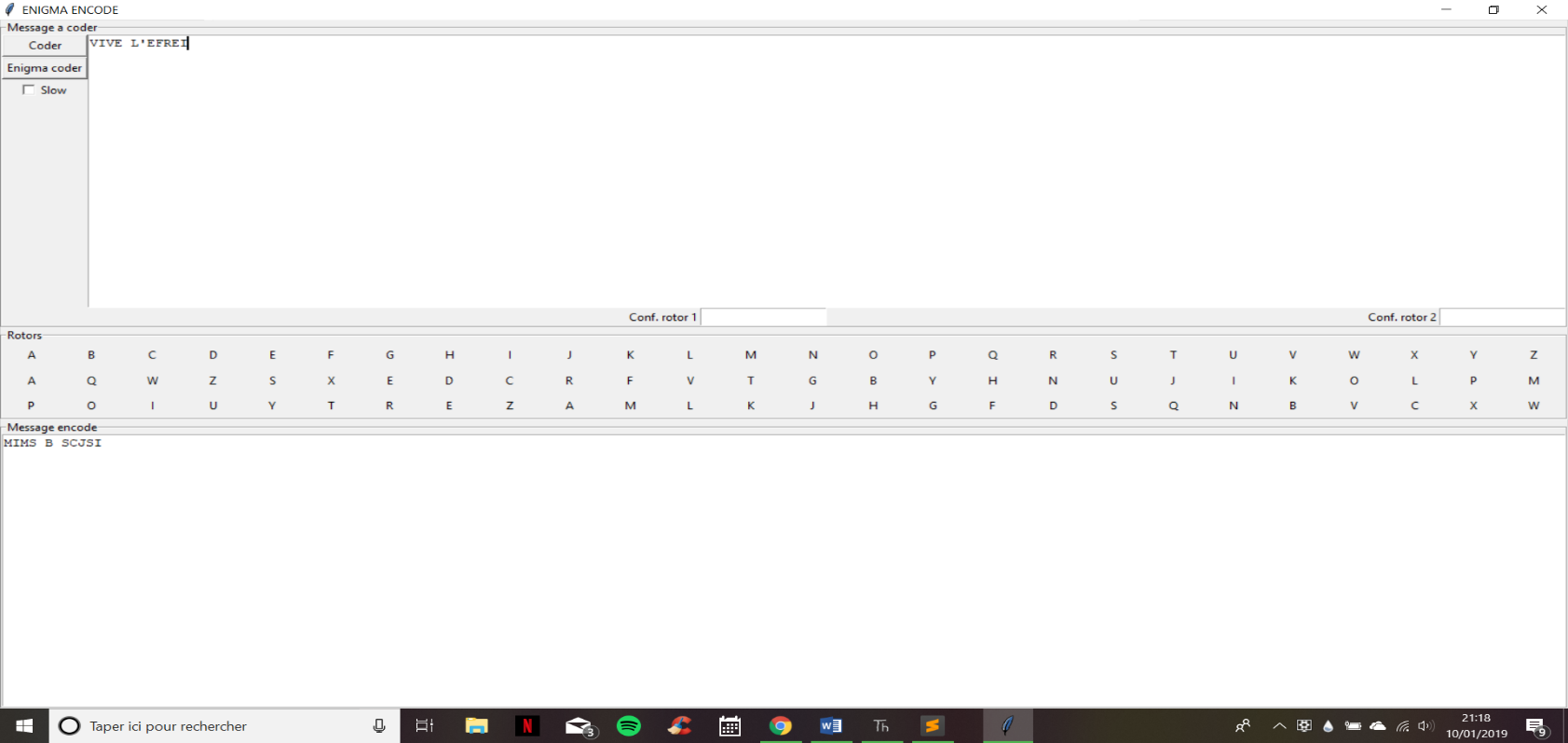
**Création de la fonction Turing**

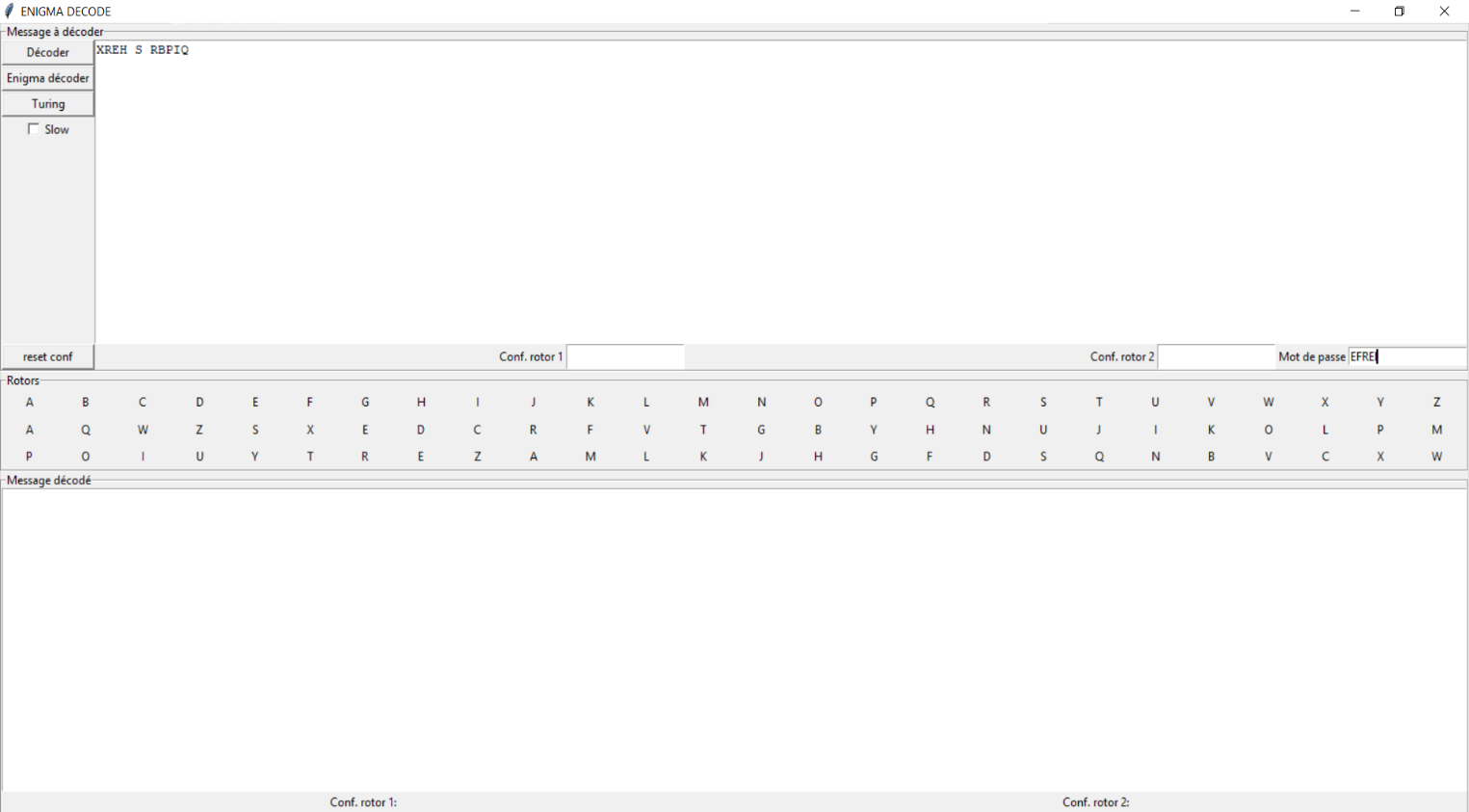
Nous avons implémenté dans cette interface graphique Enigma\_Decode un bouton « Turing » qui a été réalisé à l’aide de la fonction Button() de Tkinter.



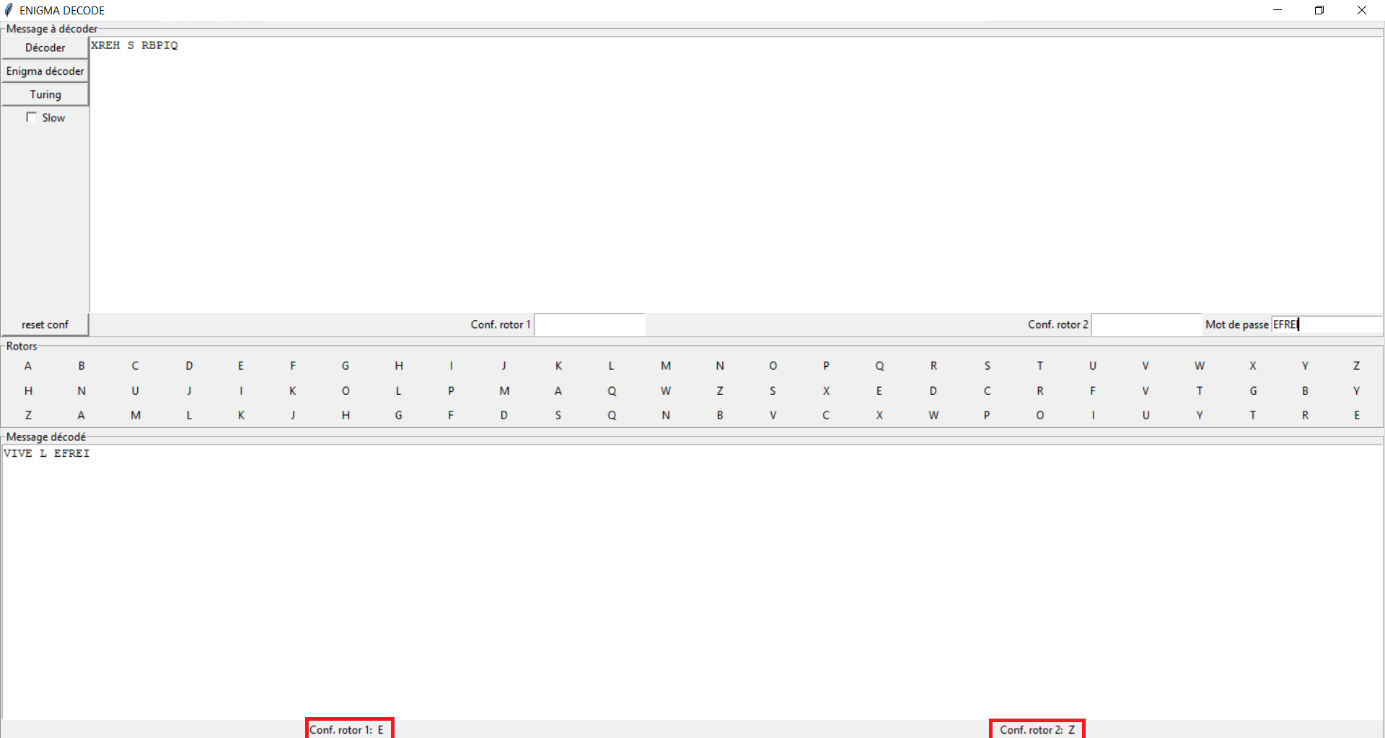
Cette fonction a pour argument la fonction turing\_enigma\_decode\_callback() qui a pour but de renvoyer les positions initiales nécessaires au décodage du message. Mais pour trouver ces positions initiales, nous avons rajouté un élément supplémentaire qui est une zone de texte nommée Mot de passe.



L’utilisateur n’aura alors plus qu’a recherché un mot étant sensé se trouvé dans le message pour trouver les positions initiales et ainsi décodé le mot.

Imaginons que nous ne sachions pas le message initial (ici « VIVE L’EFREI »), et que nous étions seulement en possession de message coder « XREH S RBPIQ » sans les conditions initiales (ici E et Z).

Il suffirait alors juste de rechercher comme Mot de passe « EFREI » ce qui permettrait de trouvé alors le message codé et de retrouvé avec quelles conditions initiales celui-ci a été codé.



On retrouve bien le message « VIVE L EFREI » (l’apostrophe n’étant pas pris en compte par le programme).  
Et on s’aperçoit que les positions initiales des rotors, E et Z, ont-elles aussi été récupérées.

**Conclusion :**

Le projet Enigma était très intéressant tant pour sa difficulté que pour sa nouveauté et son histoire. En effet cela a attisé noter curiosité de comprendre et essayer de reproduire, numériquement, la « bombe ». Ce travail nous a également permis, en plus de renforcer notre lien d’amitié et de développer de nouvelles capacités de travail en groupe : une meilleur organisation, compréhension du point de vue de l’autre, écoute et entre-aide, nous faire découvrir le fonctionnement d’une interface graphique ou Python. Ainsi, nous qui ne connaissions presque rien à Tkinter, cela fut très formateur de devoir chercher, résoudre les problèmes, essayer de faire une belle interface etc... Au niveau du code python de base, il était finalement surprenant de voir que nous avions tous deux bien progressé en 4 mois, que nous avions plus de facilité à coder et à avoir une démarche et un raisonnement de développeur. En conclusion ce projet est vraiment un excellent pour clôturer le semestre Python.

