Message au travers d'une image

Ce programme a été conçu dans l'optique d'un projet de BAC en option ISN par Simon LÉONARD et Kylian BACHELET

I. Idée et explication du projet

Le but de ce projet est de construire une image, de sorte à ce que quand on rentre un texte, il en sorte une image correspondante, et si on rentre une image, il en sorte un texte.
Pour ce faire, l'image a des spécificités particulières, comme par exemple, des valeurs permettant au programme de savoir quelle est la longueur du texte, ou comment il est `` caché '' dans l'image.

Le programme est organisé de sorte à ce que l'image et le message soient interprétés comme des objets. Cela propose une simplicité dans le cadre d'une modification du programme, on en parlera lors de la présentation des classes.

II. Organisation, répartition des tâches et parties personnelles

a. Le dépôt GIT

Dès que nous avons eu le projet en tête et que nous avons posé les bases, nous avons convenu de partager et modifier le programme via un dépôt GIT.

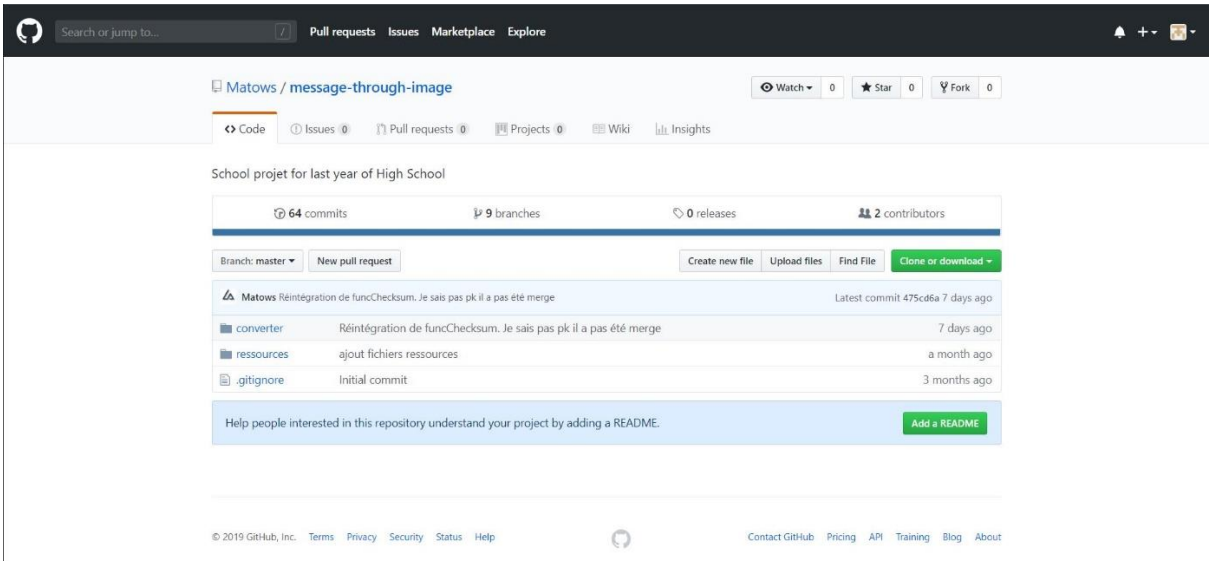
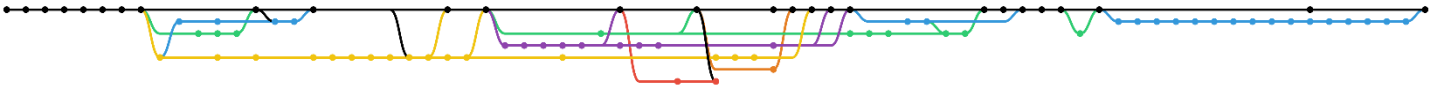


Image du dépôt GIT (https://github.com/Matows/message-through-image)

Ce dépôt nous a permis de pouvoir modifier le programme, et voir les modifications de l'autre, pour mettre en commun à la fin, et de temps à autre.

Github est un site permettant de pouvoir collaborer à plusieurs sur un même programme, tout en créant des branches. Nous nous sommes servis de ces dernières pour écrire plusieurs fonctionnalités et les modifiées séparément.



Ici, on à toutes les branches qu'on a utilisées pour ce projet, tout en haut, c'est la branche Master, celle où tout est réuni.

b. Trello

Après le dépôt GIT, on s'est aussi rapidement organisé autour de « Trello » un site permettant de faire des tableaux d'organisation.

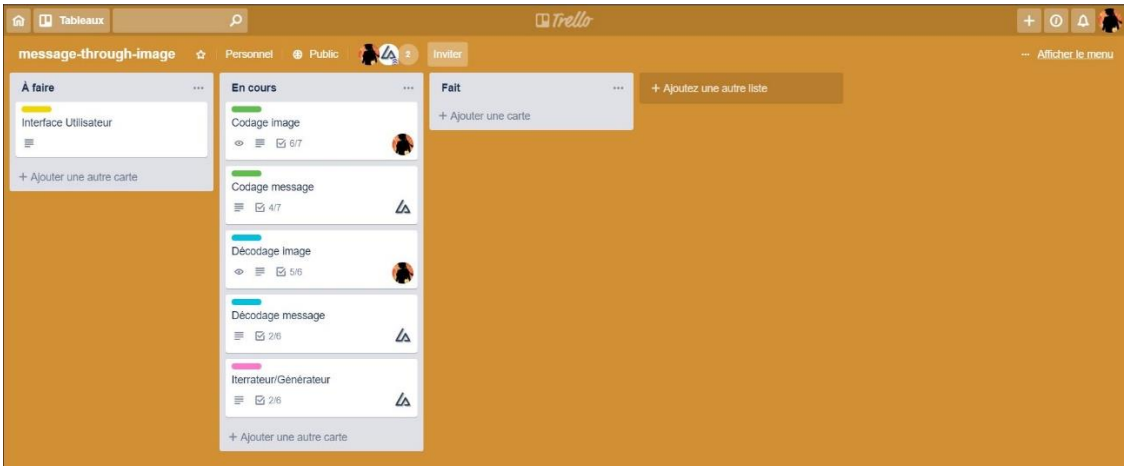
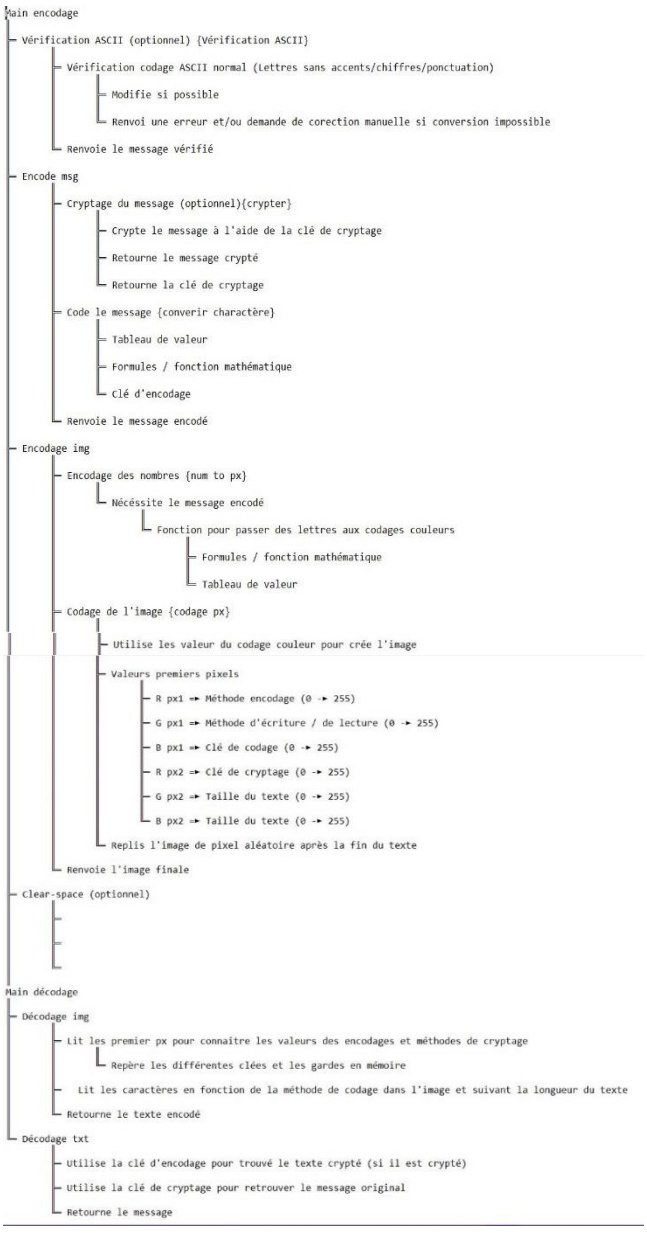


Image du Trello (https://trello.com/b/mOReGwcF/message-through-image)

Nous nous en sommes servi afin de définir qui faisait quoi et où nous en étions dans l'avancement général et particulier (c'est-à-dire dans les fonctions qu'on créait). Simon s'occupait de la partie message, et moi, de la partie image.

a. Partie personnelle de Kylian

Afin de ne pas me perdre dans ce qu'il fallait faire, j'ai décidé de me crée trois documents, le premier basique avec que du texte m'as permis de voir ce qu'on allait mettre dans chaque fonction et mieux comprendre comment fonctionne le programme.



Ce document a été légèrement modifié par la suite avec le deuxième document, qui est l'analyse fonctionnelle réalisé avec ClickChart, qui est un logiciel gratuit. Le troisième document est un organigramme qui a été réalisé avec LARP. LARP qui est originellement un logiciel de programmation pour Robot et appareils électroniques. [Tout est donné en annexe 1]

Comme tous projet, nous avons aussi eu notre lot de problème, et le plus important que j'ai eu, c'est que j'avais du mal à comprendre ce que devait faire le programme ou comment Simon voulait faire quelque chose, comme par exemple, pour l'encodage de l'image, on n'arrivait pas à se mettre d'accord sur la manière dont il fallait donner les informations.

J'ai également rencontré des problèmes avec GIT, quelques soucis de synchronisation avec les fichiers originaux, ce n'est pas grand-chose, mais ça fait perdre un peu de temps.

b. Partie personnelle de Simon

Avant de commencer à réellement coder, j'ai écrit le « squelette » de notre programme : chaque classe et chaque méthode est accompagné d'une « docstring » afin de savoir ce que fait ce bout de code.

Ce projet à été pour moi une première : j'ai découvert la programmation orienté objet, les générateurs ainsi que les branches de Git. J'ai également pu me servir des checksums pour authentifier des bouts de code.

Comme tout ce qui est nouveau, j'ai eu quelques difficultés : faire la distinction entre les attributs de classe et les variables locales, utiliser une classe dans une classe… C'est d'autant plus dur qu'il faut penser à tous les cas, même à ceux que l'on ne prévoit pas.

Mais la chose la plus compliqué à sans doute été de métriser les branches de git, et de synchroniser le tout entre plusieurs ordinateurs.

III. La gestion du programme

Nous avons décidé de faire ce projet en `` orienté objet ´´, c'est-à-dire qu'on définit le message et l'image en tant qu'objet. Comme dit précédemment, la création des classe donne un avantage sur le point de vue de la modification du programme, on peut facilement modifier une partie, sans se soucier du reste du programme. On utilise deux classes et plusieurs fonctions dans ce projet.

a. La classe Message

La classe Message est celle qui va permettre de transformer le texte en donnée utilisable pour être transformer en image. Mais il peut aussi faite l'opération inverse, c'est-à-dire qu'il peut recueillir les données de l'image pour en ressortir un message.

Elle contient les fonctions servant à vérifier les caractères ASCII, à crypter le message, à faire passer les caractères en nombre et à nettoyer le message.

b. La classe Image

La classe image, sert à crée et codée l'image selon les données que la classe Message lui donne. Il sert également à donner les données d'une image à cette même classe.

Elle contient les fonctions permettant de passer des nombre aux valeurs RGB, des valeurs RGB aux nombres, d'assembler l'image et de pouvoir la lire et extraire les valeurs RGB.

c. Les fonctions de « utils.py »

Le fichier utils.py contient diverses fonctions utiles qui permettent de vérifier que l'on ne donne que le texte ou que l'image, la fonction permettant de renvoyer un « hash md5 », et celle du générateur qui dit comment lire une image ainsi d'un algorithme de cryptage d'exemple.

IV. Comment marche le programme

Au départ, l'utilisateur rentre un message, il passe dans une fonction qui vérifie qu'il n'y a que des caractères de la table ASCII `` simple ´´ (du 0 au 84 ème [fournit en annexe]). Quand le message est correct, on demande si l'utilisateur veut le crypter, et après, il repasse dans une fonction où chaque caractère est transformé en nombre. C'est la fin de l'utilisation de la classe Message. Ensuite, c'est au tour de la classe Image. Il prend la suite de nombre que lui as donné la classe Message et la transforme en valeurs RGB. Une fois ces valeurs acquises, il les passes à une autre fonction qui va s'occuper de crée l'image. Et c'est terminé, l'utilisateur peut récupérer l'image.

Bien sûr, ça n'aurait aucun intérêt si le programme ne pouvait pas faire l'inverse. Et c'est pourquoi, on a créé les fonctions réciproques de celle pour passer du message à l'image. Dans ce cas, l'utilisateur rentre l'image, le programme en ressort les valeurs RGB qu'il va ensuite convertir en nombre. Ensuite la classe Message prend ces nombres pour les transformer en caractères, puis les décrypter si besoin et enfin, donner le message contenu dans l'image.

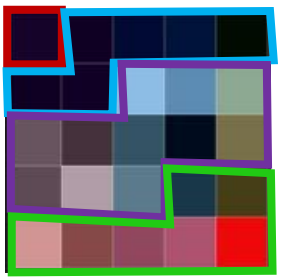
Voici une image récapitulative du rangement des données :

Pixel de « position » : défini la position du début de la chaine violette

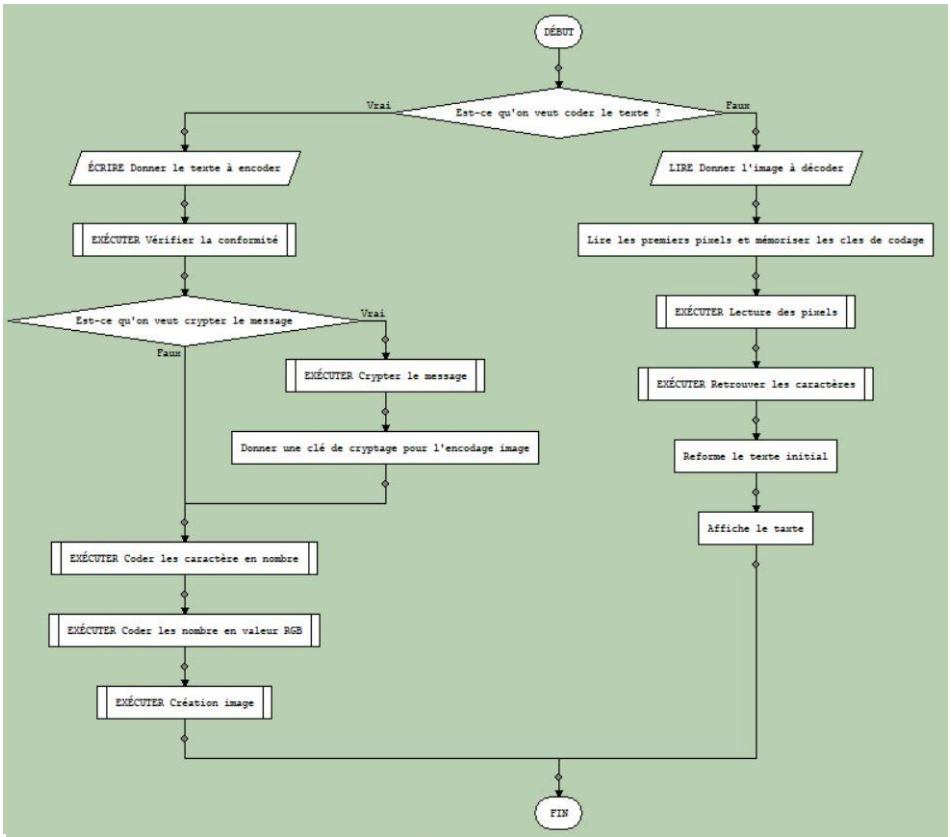
Message

Checksums

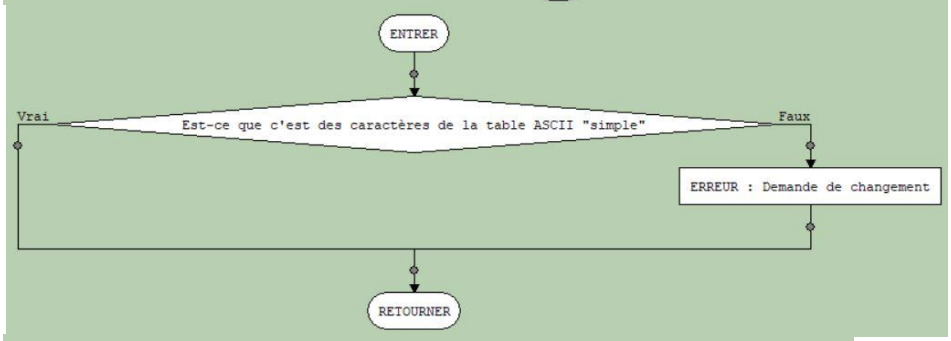
Pixels aléatoires



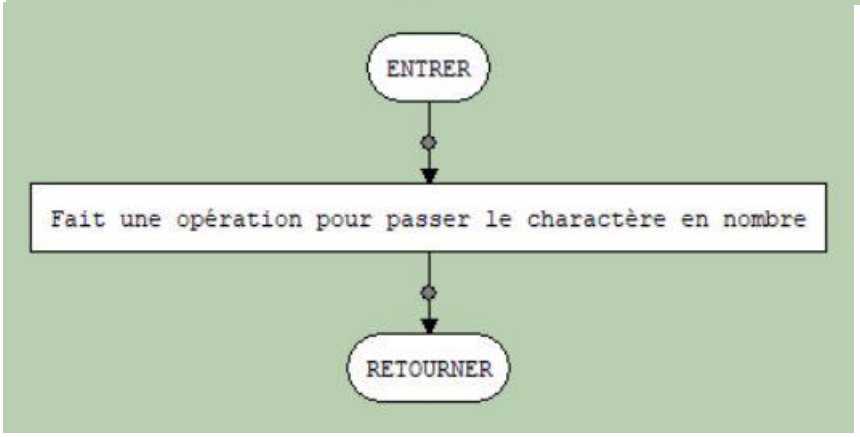
ANNEXES



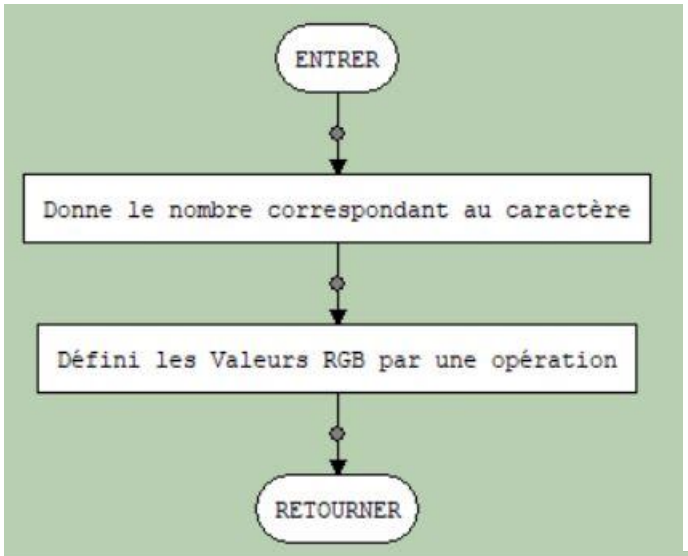
Fonctionnement du programme au complet.



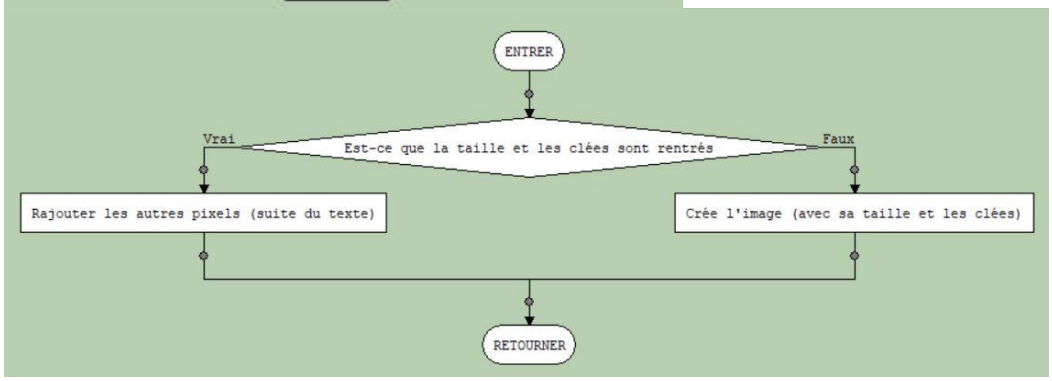
Celui-là sert à vérifier la conformité.



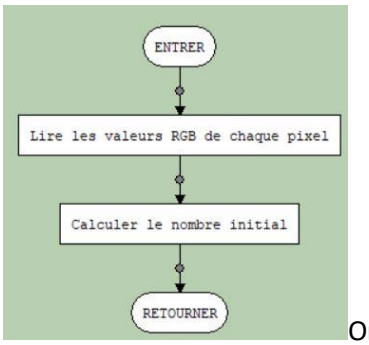
Celui-ci c'est pour passer des caractères en nombre.



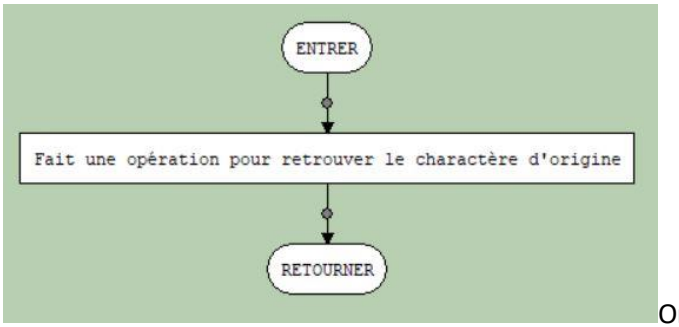
Là on passe des nombre en valeurs RGB.



Ici c'est pour créer l'image.

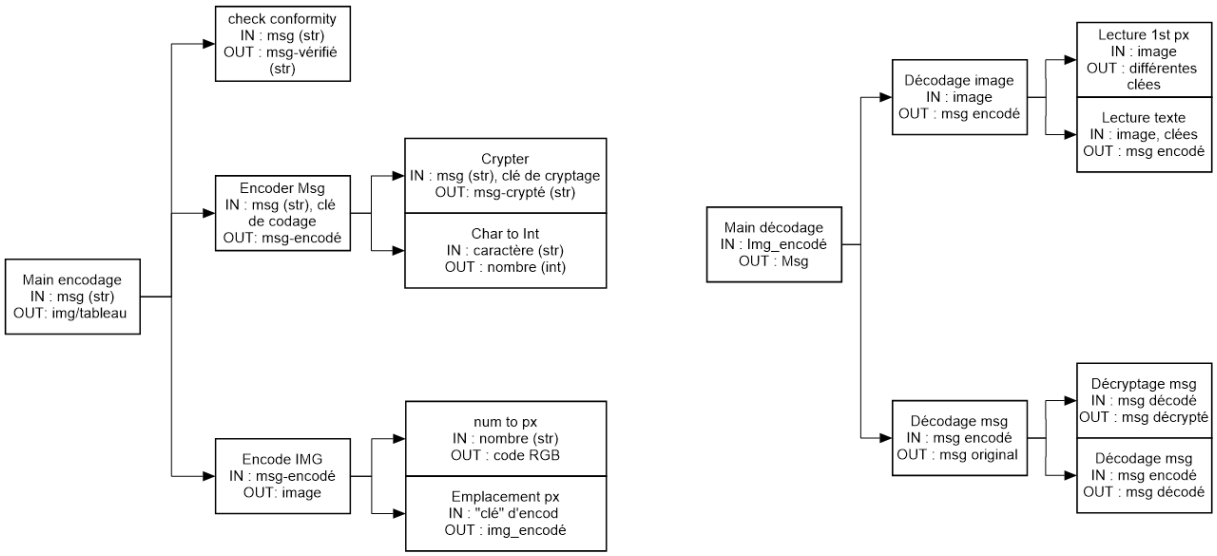


On lit les valeurs RGB.



On retrouve les nombres.

L'analyse fonctionnelle :



Encodage :

Inputs