Conception d'un protocole de communication graphique

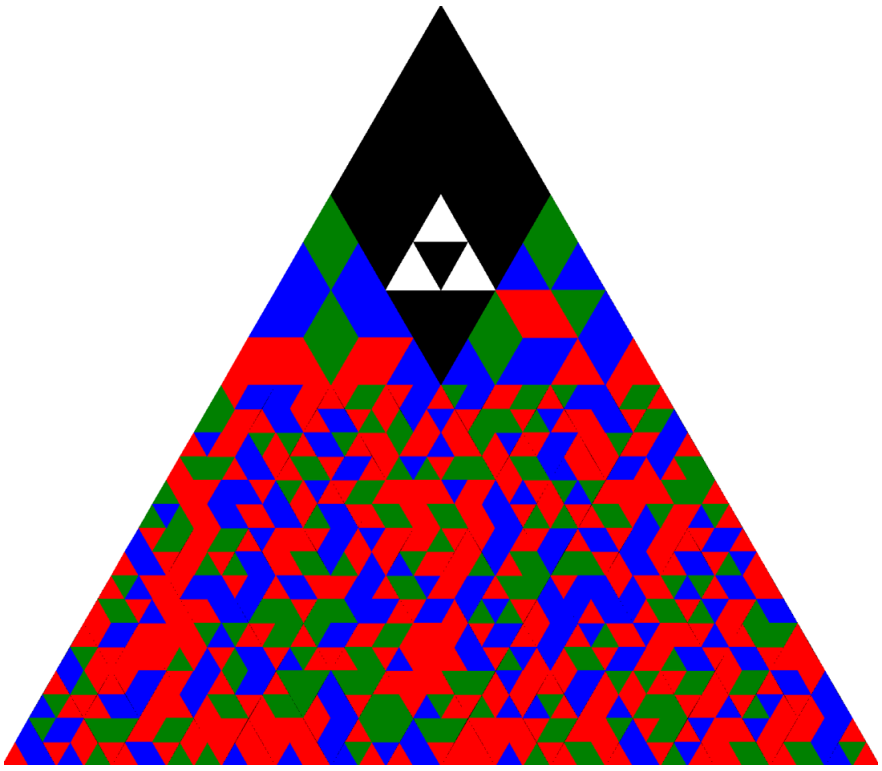
EL AMRANI Wadie
BAHOU Yassine

Table des matières

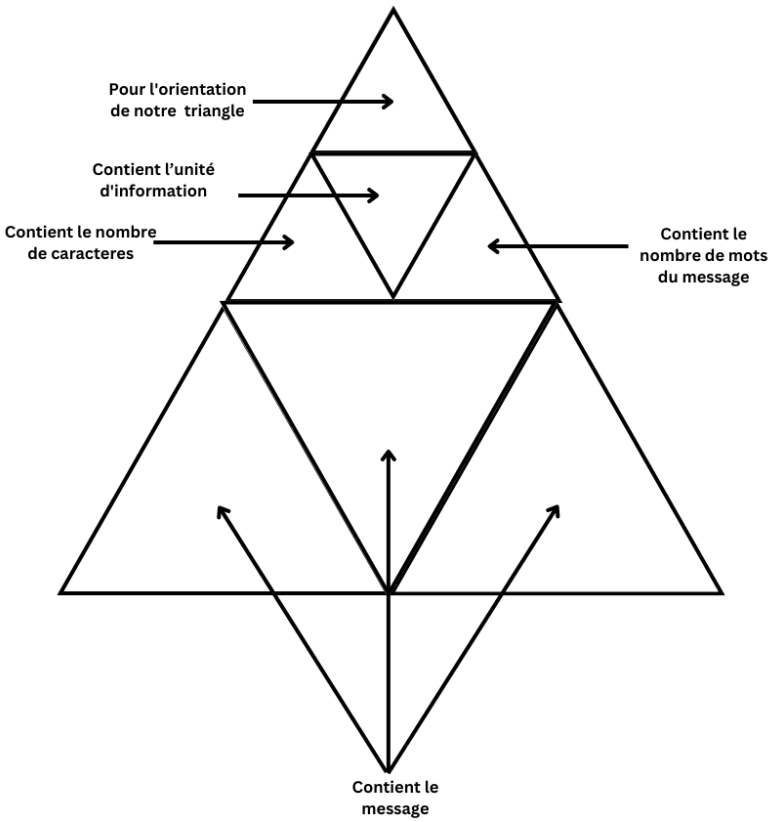
1 Introduction

Dans le cadre de ce TP, nous sommes amenés à concevoir un protocole de communication graphique. L'objectif est de proposer une méthode de transmission d'informations visuelles qui puisse être imprimée et lue par une machine. Un exemple de protocole graphique connu est le QR code, qui permet de stocker une grande quantité d'informations dans un petit espace.

Notre tâche consiste à concevoir un nouveau protocole de A à Z en proposant des solutions originales aux problèmes posés. Nous devrons dessiner les informations dans une matrice constituée de cellules, où chaque cellule représente une unité d'information de base. Nous devrons également inclure des informations permettant de détecter et de corriger des erreurs, ainsi que des informations spécifiques à notre protocole.



2 Idée génerale



Pour notre QR code, nous avons décidé d'utiliser des triangles équilatéraux comme unité de base. Nous avons dessiné un grand triangle équilatéral qui est divisé en quatre sous-triangles équilatéraux. Le triangle du haut contient des informations supplémentaires au message, et il est lui-même divisé en quatre sous-triangles. Le triangle du haut est totalement colorié en noir, tandis que les trois autres contiennent respectivement le nombre de caractères, le nombre de mots et un petit triangle noir qui désigne notre cellule d'informations.

Les trois autres triangles en bas contiennent notre message codé. L'idée générale est d'effectuer un pavage de triangles en petits triangles pour représenter les informations de manière compacte et efficace.

Nous pensons que cette méthode permettra de stocker une grande quantité d'informations dans un petit espace tout en facilitant la lecture et la détection d'erreurs. Nous expliquerons plus en détail notre idée dans la suite.

3 Implémentation

Pour la réalisation de ce projet, nous avons choisi d'utiliser le langage de programmation Python. Ce choix s'est imposé pour plusieurs raisons. Tout d'abord, Python est un langage dynamique qui permet une programmation rapide et efficace. De plus, il est facile à coder et à lire, ce qui facilite le travail en équipe et la maintenance du code.

Ensuite, pour la manipulation d'images, nous avons utilisé la bibliothèque Pillow. Cette bibliothèque est très complète et permet de manipuler des images de différents formats (JPEG, PNG, BMP, etc.). Elle nous a permis de facilement créer et manipuler des images en Python. De plus, elle est très bien documentée et dispose d'une grande communauté, ce qui nous a facilité la prise en main.

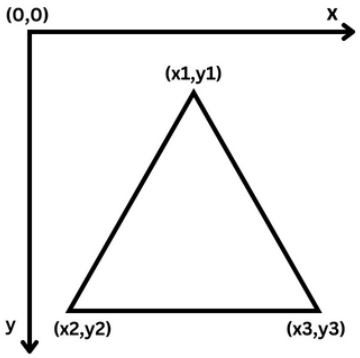
Enfin, Python étant un langage de programmation de haut niveau, nous avons pu éviter les erreurs de types et les problèmes de gestion de la mémoire, ce qui nous a permis de nous concentrer sur l'implémentation de notre algorithme.

En somme, le choix de Python et de la bibliothèque Pillow s'est révélé être une excellente décision pour ce projet, nous permettant d'implémenter notre algorithme de manière efficace et sans problème majeur.

3.1 Fonction triangle

La fonction `triangle(size)` dessine un triangle équilatéral noir de taille `size` et renvoie les coordonnées des sommets ainsi que l'image créée.

[ ]:
```python
def triangle(size):
    height = int(math.sqrt(3) * size)
    width = size * 2
    x1 = width / 2
    y1 = 0
    x2 = 0
    y2 = height
    x3 = width
    y3 = height
    image = Image.new('RGB', (width, height), color='white')
    draw = ImageDraw.Draw(image)
    draw.polygon([(x1, y1), (x2, y2), (x3, y3)] , fill='black')
    return [(x1, y1), (x2, y2), (x3, y3)], image
```



Les variables utilisées sont :

- `size` : la taille du triangle
- `height` : la hauteur du triangle calculée à partir de la taille
- `width` : la largeur du triangle calculée à partir de la taille
- `x1`, `y1`, `x2`, `y2`, `x3`, `y3` : les coordonnées des sommets du triangle
- `image` : l'image créée pour dessiner le triangle
- `draw` : l'objet `ImageDraw` utilisé pour dessiner sur l'image

3.2 Fonction draw and get sub triangles

```
[ ]: def draw_and_get_sub_triangles(points, size, image):
        x1, y1 = points[0]
        x2, y2 = points[1]
        x3, y3 = points[2]

        height = int(math.sqrt(3) * size)
        width = size * 2

        draw = ImageDraw.Draw(image)

        # Draw the main triangle
        draw.polygon([(x1, y1), (x2, y2), (x3, y3)], fill='black')

        # Calculate the sub-triangles
        triangle1 = [(x1, y1), ((x1+x2)/2, (y1+y2)/2), ((x1+x3)/2, (y1+y3)/2)]
        triangle2 = [((x1+x2)/2, (y1+y2)/2),(x2, y2), ((x2+x3)/2, (y2+y3)/2)]
        triangle3 = [((x1+x3)/2, (y1+y3)/2), ((x2+x3)/2, (y2+y3)/2), (x3, y3)]
        triangle4 = [((x2+x3)/2, (y2+y3)/2),((x1+x2)/2, (y1+y2)/2),((x1+x3)/
    ↪2, (y1+y3)/2)]
        # Draw the sub-triangles
        draw.polygon(triangle1, fill='red')
        draw.polygon(triangle2, fill='green')
        draw.polygon(triangle3, fill='blue')
        draw.polygon(triangle4, fill='yellow')



        return [triangle1, triangle2, triangle3, triangle4]
```

Cette fonction prend en entrée trois arguments :

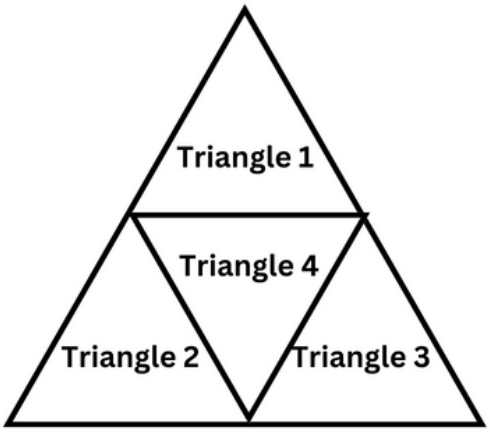
- **points** : une liste de trois tuples représentant les coordonnées des sommets du triangle à diviser.
- **size** : un entier représentant la taille de l'image .
- **image** : une image Pillow sur laquelle dessiner les triangles.

La fonction calcule les quatre sous-triangles en utilisant les coordonnées des sommets du triangle initial. Chaque sous-triangle est défini par une liste de trois tuples représentant

ses coordonnées. Ces sous-triangles sont stockés dans des variables nommées `triangle1`, `triangle2`, `triangle3` et `triangle4`.

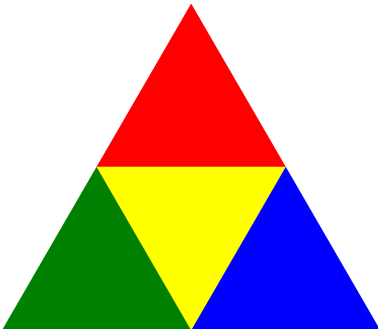
Enfin, la fonction dessine les quatre sous-triangles en utilisant la méthode `polygon` de l'objet `ImageDraw.Draw`. Chaque sous-triangle est dessiné avec une couleur différente (rouge, vert, bleu et jaune).

La fonction renvoie une liste contenant les coordonnées des sommets des quatre sous-triangles.



3.2.1 Test

[ ]:
```
size = 750
points,image=triangle(size)
triangles_1 = draw_and_get_sub_triangles(points, size, image)
image.show()
```



3.3 Codage de caractères avec un dictionnaire

```
[ ]: def codage_dict(number_of_colors =  3):
    # we can augment the number of colors to increase
    # the weight of hamming by incresing the distance between charcters␣
 ↪whichs alowed since the
    # the number of available combination is greater then the number of␣
 ↪charcters
    # ex : 3 colors ====>  81 combination
    #      4 colors ====> 256 combination
    #      5 colors ===> 625 combination
    colors = ['red','green','blue','yellow','white','black']
    dict_num_colors = {}
    for i in range(len(colors)):
        dict_num_colors[i]=colors[i]
    charcters =␣
 ↪"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 ?:/.!
 ↪*-+"
    dict_char_num ={}
    pt = 0
    while pt <len(charcters):
        for i in range(number_of_colors-1,-1,-1):
            for j in range(number_of_colors-1,-1,-1):
                for k in range(number_of_colors-1,-1,-1):
                    for l in range(number_of_colors-1,-1,-1):
                        if (i,j,k,l) not in dict_char_num.values():
                            dict_char_num[charcters[pt]]=(i,j,k,l)
        pt +=1
    dict_char_num["@"] = (5,5,5,4)
    return dict_num_colors,dict_char_num
```

Le `codage` consiste à représenter un ensemble de caractères par une suite de codes. Dans la fonction `codage_dict`, nous utilisons un dictionnaire pour encoder les caractères. Le dictionnaire est composé de deux parties : `dict num colors` et `dict_char_num`.

`dict num colors` est un dictionnaire qui associe un nombre à chaque couleur. Les couleurs sont définies dans la liste `colors` et peuvent être augmentées pour renforcer la distance entre les caractères.

`dict_char_num` est un dictionnaire qui associe un caractère à une suite de nombres correspondant aux codes de couleurs. Le nombre de couleurs disponibles pour le codage est défini par le paramètre `number_of_colors`. La fonction utilise quatre boucles imbriquées pour générer toutes les combinaisons de codes de couleurs possibles. Si la combinaison est déjà présente dans le dictionnaire, elle est ignorée. La boucle externe parcourt la liste `characters` pour ajouter chaque caractère et son code au dictionnaire `dict_char_num`. Une entrée spéciale pour le caractère @ est ajoutée avec un code spécifique (5,5,5,4).

Enfin, la fonction retourne les deux dictionnaires `dict_num_colors` et `dict_char_num`. Ceux-ci peuvent être utilisés pour encoder et décoder des messages à l'aide des codes de couleurs associés aux caractères.

3.4 Codage de caractères en triangles

Le nombre de triangles que l'on peut diviser dépend directement de la taille du triangle initial, qui est déterminée par les points de départ. La fonction `draw_message_16()` divise un triangle initial en quatre triangles à chaque itération, ce qui permet de créer un maximum de 16 caractères dans chaque 3 triangles, donc $16 \times 3 = 48$, ce qui signifie que la taille maximale du message qu'on peut envoyer est 48. D'autre part, la fonction `draw_message_64()` divise un triangle initial en quatre triangles à chaque itération, et chaque triangle est lui-même divisé en quatre triangles permettant ainsi de créer un maximum de 64 caractères dans chaque triangle, donc $64 \times 3 = 192$ est la taille maximale , ce qui suffisant pour nous dans ce TP.

Plus la division est grand, plus la taille maximale du message que l'on peut encoder est grande. Cependant, cela dépend également de la capacité de stockage des caractères utilisés pour encoder le message, et donc de la précision et de la taille de la matrice d'images utilisée pour stocker le message encodé.

[1]:
```
def draw_message_16(points, size, image,msg):
    dict_num_colors,dict_char_num = codage_dict()
    triangles_1 = draw_and_get_sub_triangles(points, size, image)
    pt = 0
    for i in range(4):
        tmp = draw_and_get_sub_triangles(triangles_1[i], size, image)
        for j in range(4):
            tmp1 =  draw_and_get_sub_triangles(tmp[j], size,␣
↪image,dict_char_num[msg[pt]])
            pt += 1
    image.show()
```

[ ]:
```
def draw_message_64(points, size, image,msg):
    dict_num_colors,dict_char_num = codage_dict()
    triangles_1 = draw_and_get_sub_triangles(points, size, image)
    pt = 0
    for i in range(4):
        tmp = draw_and_get_sub_triangles(triangles_1[i], size, image)
        for j in range(4):
            tmp1 =  draw_and_get_sub_triangles(tmp[j], size, image)
            for k in range(4):
                tmp2 =  draw_and_get_sub_triangles(tmp1[k], size,␣
↪image,dict_char_num[msg[pt]])
                pt += 1
    image.show()
```

Ensuite, pour avoir un triangle homogène, on a utilisé une fonction qui regroupe le message en blocs de 64 caractères par cette fonction :

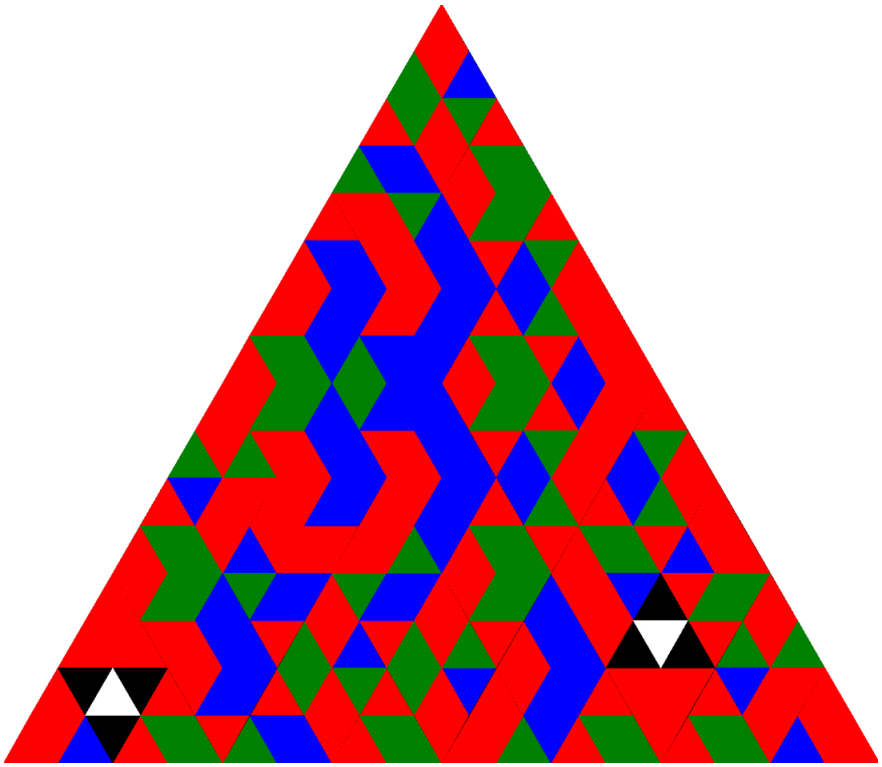
[1]:
```
def make_it(s,num):
    #this function makes a string have 16 chars by duplicationg it after␣
↪adding @
    # '123' ===>'123@123@123@123@'
    s += "@"
    while len(s) < num:
        s += s
    return s[:num]
```

La fonction qui permet donc de creer le triangle associe au message :

[ ]:
```
def draw_msg_under_64(points, size, image,msg,num):
    msg = make_it(msg,num)
    draw_message_64(points, size, image,msg)
```

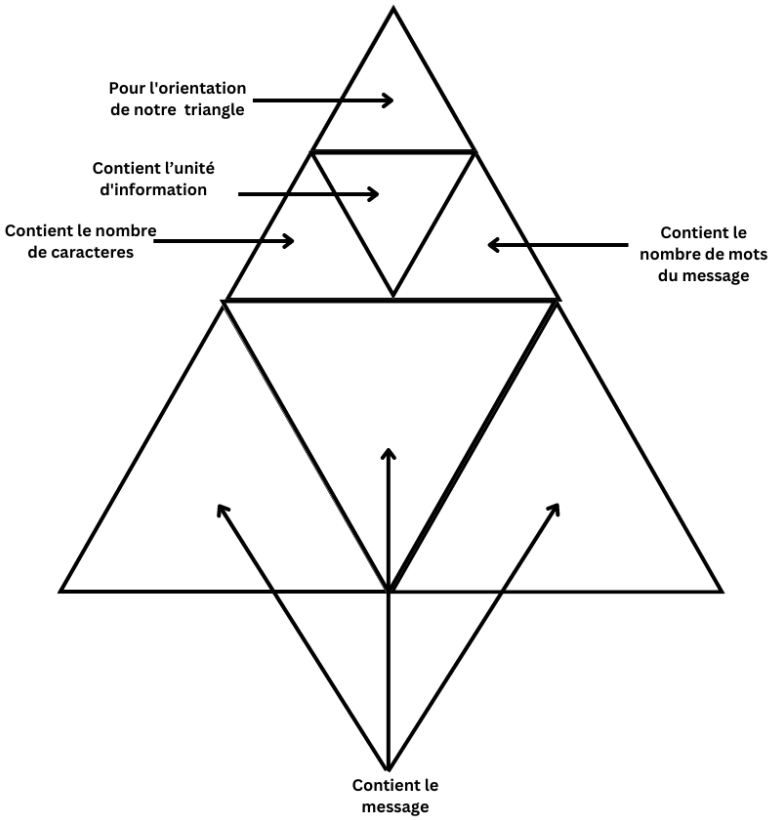
3.4.1 Test message

[1]:
```
size = 750
points,image=triangle(size)
num = 16 * 4
msg="premier test de message"
draw_msg_under_64(points, size, image,msg,num)
image.show()
```



3.5 La fonction principale

3.5.1 Les informations sur le message



En effet, la forme de notre triangle doit respecter la conception definie dans la premiere section , le premier triangle en haut doit inclure les informations sur le message. On divise donc le message en 3 sous message , et les coder sur les 3 triangle en bas
Ces fonctions calculent ne nombre de caracteres et le nombre de mots d'un message :

[1]:
```
def decompose_to_3(msg):
    if len(msg)%3==0:
        tmp = len(msg)//3
    else:
        tmp = len(msg)//3+1
    return [msg[0:tmp],msg[tmp:2*tmp],msg[2*tmp:]]
def codage_taille(taille):
    dict_num_colors,dict_char_num = codage_dict()
```

```
    msg_size = str(taille)
    while len(msg_size)<4:
        msg_size = "0"+msg_size
    tmp = []
    for chiffre in msg_size:
        tmp.append(dict_char_num[chiffre])
    return tmp
def draw_number(points, size, image,number):
    tmp = codage_taille(number)
    tout = draw_and_get_sub_triangles(points, size, image,(5,5,5,5))
    i = 0
    for chiffre in tmp:
        draw_and_get_sub_triangles(tout[i], size, image,chiffre)
        i +=1
def count_words(string):
    # Split the string into a list of words
    words = string.split()
    # Return the length of the list
    return len(words)
```

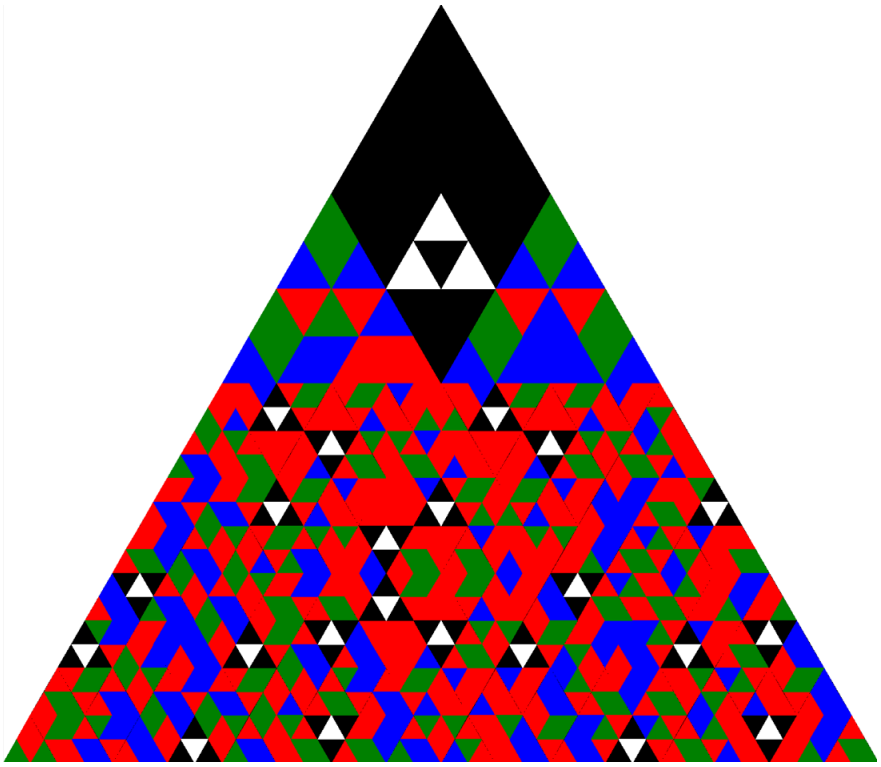
3.5.2 Le code Main du codage

[1]:
```
def main(points, size, image,msg):
    dict_num_colors,dict_char_num = codage_dict()
    triangles_1 = draw_and_get_sub_triangles(points, size, image)
    # -------------protocole information-------------
    tmp = draw_and_get_sub_triangles(triangles_1[0], size, image)
    # orientation black head
    tmp1 = draw_and_get_sub_triangles(tmp[0], size, image,(5,5,5,5))
    # number of characters
    taille = len(msg)
    draw_number(tmp[1], size, image,taille)
    # number of words
    words = count_words(msg)
    draw_number(tmp[2], size, image,words)
    # triangle size
    tmp3 = draw_and_get_sub_triangles(tmp[3], size, image,(5,5,5,5))
```

```
    tmp4 =  draw_and_get_sub_triangles(tmp3[3], size, image,(4,4,4,4))
    tmp5 =  draw_and_get_sub_triangles(tmp4[0], size, image,(4,4,4,4))
    tmp5 =  draw_and_get_sub_triangles(tmp4[1], size, image,(4,4,4,4))
    tmp5 =  draw_and_get_sub_triangles(tmp4[2], size, image,(4,4,4,4))
    tmp5 =  draw_and_get_sub_triangles(tmp4[3], size, image,(5,5,5,5))
     # -------------message-------------
    list_msg = decompose_to_3(msg)
    for i in range(len(list_msg)):
        draw_msg_under_64(triangles_1[i+1], size, image,list_msg[i],64)
```

3.5.3 test de la fonction Main

[1]:
```
size = 750
points,image=triangle(size)
msg="premier test de message"
main(points, size, image,msg)
image.show()
```



[1]:
```
size = 750
points,image=triangle(size)
```

```
msg="Le TP de reseau etait vraiment interessant ! On a realise un␣
 ↪protocole de communication graphique en utilisant le pavage des␣
 ↪triangles et seulement 4 couleurs. C'etait une experience fascinante␣
 ↪qui nous a permis de mieux comprendre les concepts de base de la␣
 ↪communication reseau."
main(points, size, image,msg)
image.show()
```