Réseaux Mobiles et Sans Fil : Deux Devoirs

Réalisé par :

Mehdi CHARIFE

Destiné à :

Prof. Amine BERQIA

Année universitaire : 2022-2023

Table des matières

Table des figures

Chapitre 1

Normes IEEE 802.11

1.1 Mise en perspective

La connectivité sans fil est devenue un élément crucial de notre vie quotidienne, permettant aux utilisateurs de rester connectés à Internet où qu'ils soient. Cependant, cela n'a pas été toujours le cas. Avant l'arrivée du standarisation, les réseaux locaux sans fil étaient largement fragmentés, avec de nombreux protocoles propriétaires utilisés par différents fabricants. Pour remédier à cela, l'organisation IEEE a élaboré le standard IEEE 802.11 qui permettrait une interopérabilité entre les appareils réalisés par différents fabricants, facilitant ainsi la connectivité sans fil pour les utilisateurs finaux.

1.2 Présentation des normes

IEEE 802.11 est un ensemble de normes pour les réseaux locaux sans fil développé par le comité des normes IEEE LAN/MAN. Ces normes spécifient un ensemble de caractéristiques techniques au niveau du matériel et des logiciels des appareils sans fil, qui sont utilisés pour créer, maintenir, et exploiter des réseaux sans fil.

1.3 Générations

Les normes 802.11 ont été régulièrement améliorées depuis leur création en 1997 par l'IEEE. Ces améliorations sont appelées des amendements à la norme initiale et sont gérés et approuvés par le groupe de travail 802.11 de l'IEEE. La principale utilisation commerciale de ces normes est la technologie Wi-Fi.

Le tableau suivant résume quelques différence entre ces protocols.

Protocole	Sortie	Fréquence	Débit binaire (Mbits/s)	Portée	
				Int.	Ext.
802.11 1997	1997	2.4 GHz	1,2	20m	100m
802.11a	1999	5 GHz	6, 9, 12, 18, 24, 36, 48, 54	35m	120m
802.11b	1999	2.4 GHz	1, 2, 5.5, 11	25m	140m
802.11g	2003	2.4 GHz	6, 9, 12, 18, 24, 36, 48, 54	38m	140m

Chapitre 2

Ré-utilisation de fréquence

Le but de cette partie est de concevoir et réaliser une API qui, à partir de deux entiers i et j et un réel r, dessine un motif d'hexagones réguliers de rayon r dont les cellules ayant la même fréquence sont montrées avec la meme couleur.

2.1 Approche utilisée

La stratégie adoptée était de commencer avec un seul hexagone, que l'on note H_0, et d'essayer d'exprimer les coordonnées des 6 hexagones dont la distance de H_0 permet une ré-utilisation de fréquence en fonction de i, j, r et les coordonnées de H_0.

Étant qu'il existe une symètrie vis-à-vis le centre de H_0 entre les centres des hexagones au dessous et au dessus de H_0, il est suffit d'exprimer les coordonnées des hexagones supérieurs pour déduire ceux des hexagones inférieurs.

Pour l'hexagone dans l'extrémité droite supérieure, que l'on note par la suite H_1, on a constaté que les coordonnées peuvent s'exprimer ainsi :

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 + (3 + 1.5(i-2))r \\ y_0 + \sqrt{3}(j + 0.5i)r \end{pmatrix} \tag{2.1}$$

où x_0 et y_0 sont les coordonnées du centre de H_0.

Ainsi, pour trouver les coordonnées x_4 et y_4 de l'hexagone à l'extrémité gauche inférieure, il suffit de multiplier les coordonnées de H_1 par -1 :

$$\begin{pmatrix} x_4 \\ y_4 \end{pmatrix} = \begin{pmatrix} -x_1 \\ -y_1 \end{pmatrix} \tag{2.2}$$

Concernant les deux hexagones supérieurs H_2 et H_3, l'exploitation de la figure **2.1** réalisée par l'outil `GeoGebra` a permet d'aboutir à la formule générale suivante :

$$\begin{cases} x_{n+1} = x_n - \sqrt{3(i^2+ij+j^2)}r\cos\left(-\pi/6 + \arcsin\left(x_n - x_0\right)\right) \\ \\ y_{n+1} = y_n + \sqrt{3(i^2+ij+j^2)}r\sin\left(-\pi/6 + \arcsin\left(x_n - x_0\right)\right) \end{cases}$$

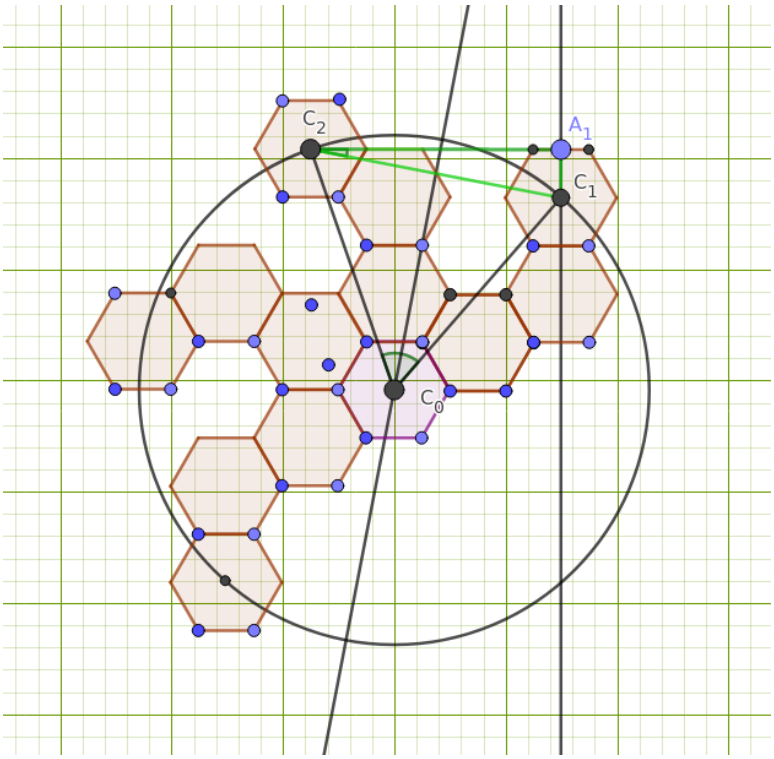

Figure 2.1 – $i = 2$, $j = 1$

Preuve :

D'après la figure ci-dessus, on peut constater que, dans le triangle $C_0C_1C_2$:

$$\cos(C_2\hat{C}_0C_1) = \frac{x_n - x_{n+1}}{\sqrt{3(i^2+ij+j^2)}r} \qquad \sin(C_2\hat{C}_0C_1) = \frac{y_{n+1} - y_n}{\sqrt{3(i^2+ij+j^2)}r}$$

Donc :

$$x_{n+1} = x_n - \sqrt{3(i^2 + ij + j^2)}r\cos\left(C_0\hat{C_1}C_2\right)$$

$$y_{n+1} = y_n + \sqrt{3(i^2 + ij + j^2)}r\sin\left(C_0\hat{C_1}C_2\right)$$

Par contre :

$$C_2\hat{C_0}C_1 = \frac{\pi}{2} - (\pi - (\frac{\pi}{3} + \arcsin\left(\frac{x_n - x_{n+1}}{\sqrt{3(i^2 + ij + j^2)}r}\right)))$$

$$C_2\hat{C_0}C_1 = -\frac{\pi}{6} + \arcsin\left(\frac{x_n - x_{n+1}}{\sqrt{3(i^2 + ij + j^2)}r}\right)$$

Ainsi :

$$\begin{cases} x_{n+1} = x_n - \sqrt{3(i^2 + ij + j^2)}r\cos\left(-\pi/6 + \arcsin\left(x_n - x_0\right)\right) \\ \\ y_{n+1} = y_n + \sqrt{3(i^2 + ij + j^2)}r\sin\left(-\pi/6 + \arcsin\left(x_n - x_0\right)\right) \end{cases}$$

■

Équipés de ces réalisations, une API utilisant les formules évoqués ci-dessus a été mise en place pour dessiner un motif de cellules dont ceux ayant la même fréquence sont montrées avec la même couleur.

2.2 Mise en oeuvre

2.2.1 Choix de la technologie

Pour l'implémentation de l'`API`, le verdict est tombé sur la bibliothèque `matplotlib` du language *Python*. Cette bibliothèque permet de créer des graphiques de haute qualité dans un large éventail de formats. Elle est souvent utilisée pour créer des graphiques scientifiques et des graphiques d'analyse de données. Elle offre une grande flexibilité pour personnaliser les graphiques en termes de style, de couleurs et de mise en page. Elle peut être utilisé pour tracer des courbes, des histogrammes, des diagrammes à barres, des nuages de points, des surfaces 3D, des cartes et bien plus encore. Elle est largement utilisée dans les domaines de la science, de l'ingénierie, de la finance et de la recherche.

Figure 2.2 – Logo de la bibliothèque `matplotlib`

Particulièrement, et nécéssaire à notre cas, `matplotlib` permet de tracer des formes géométriques de base telles que des hexagones avec une grande précision, en utilisant des fonctions telles que `polygon()` ou `RegularPolygon()`. De plus, il fournit des fonctionnalités pour organiser les formes sur la surface de dessin en utilisant des sous-graphiques, des couches et des projections. Enfin, il offre une grande flexibilité pour personnaliser les formes tracées en termes de style, de couleurs et de mise en page, ce qui est essentiel pour créer un motif complexe et esthétiquement agréable.

2.2.2 Découpage

L'API réalisée est composée de deux parties principales : Une partie qui dessine le motif des cellules et une partie qui se charge de colorer les cellules ayant la méme fréquence. La première partie est de son tour divisée en deux sous-parties dont chacune s'occupe de dessiner une des parties inférieures et supérieures du motif.

2.2.2.1 Partie du Dessin :

```
def draw(x,y,radius,n):
    drawTop(x,y,radius,n)
    drawBottom(x,y,radius,n)
```

La fonction `draw()` fait appel aux deux fonctions `drawTop()` et `drawBottom()` qui dessinent les parties supérieures et inférieures du motif respectivement :

```
def drawTop(x,y,r,n):
    k = n
    drawTopLeft(x,y,r,k)
    drawTopRight(x,y,r,n)
```

```
def drawBottom(x,y,r,n):
    drawBottomRight(x,y,radius,n)
    drawBottomLeft(x,y,radius,n)
```

Chacune des deux fonctions fait appel à d'autres fonctions qui se chargent de dessiner les partie gauches et droites correspondantes. Par exmple, `drawTop()` fait appel aux fonctions `drawTopLeft()` et `drawTopRight()`.

Le reste du partie du dessin peut être consulté dans l'annexe contenant la totalité du code source.

2.2.2.2 Partie du colorage :

```
def color(x,y,radius,i,j,n):
    x0, y0 = x1, y1
    distance = np.sqrt(3*(i**2 + i*j + j**2))*radius
    hexagon = mpatches.RegularPolygon((x0, y0), 6,
                radius=radius, orientation=np.pi/6,
                facecolor='#ffaadf', linewidth=1
    )
    ax.add_patch(hexagon)

    x1 = (3+(i-2)*3/2)*radius + x0
    y1 = (j + i/2)*np.sqrt(3)*radius + y0
    for k in range(0,3):
        for p in range(0,2):
            hexagon = mpatches.RegularPolygon(
                        (2*p*x0 + ((-1)**p)*x1, 2*p*y0 +
                        ((-1)**p)*y1),
                        6,
                        radius=radius, orientation=np.pi/6,
                        facecolor='#3c001b', linewidth=1
            )
            ax.add_patch(hexagon)


        a = x1
        x1 = x1 - 2*distance*np.sin(np.pi/6)*np.cos(-np.pi/6 +
        np.arcsin((x1-x0)/distance))
        y1 = y1 + 2*factor*np.sin(np.pi/6)*np.sin(-np.pi/6 +
        np.arcsin((a-x0)/distance))



    ax.set_xlim(-10.5*radius, 10.5*radius)
    ax.set_ylim(-10.5*radius, 10.5*radius)
    plt.show()
```

2.3 Démonstrations

2.3.1 $i = 2, j = 1$

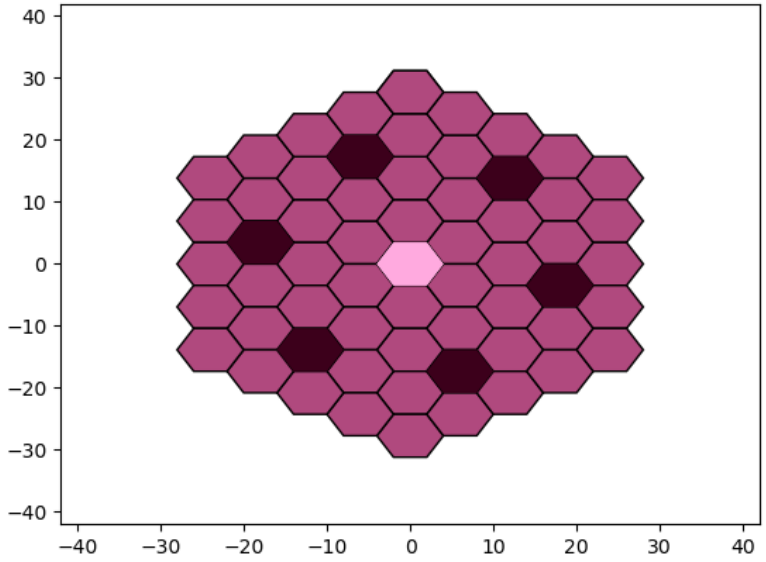

Figure 2.3 – $i = 2, j = 1$

2.3.2 $i = 3, j = 2$

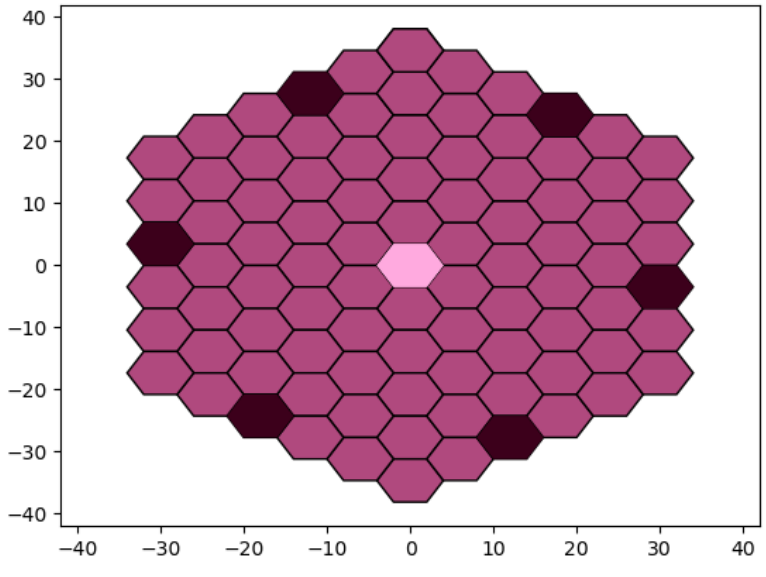

Figure 2.4 – $i = 3, j = 2$

2.3.3 $i = 5, j = 12$

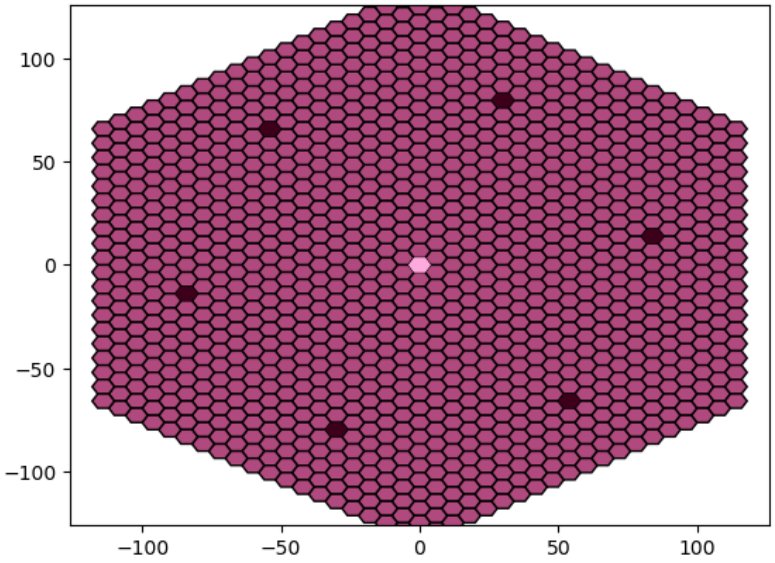

Figure 2.5 – $i = 5, j = 12$

Annexes

Annexe A

Code source

A.1 draw.py

Modules importés

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import numpy as np
```

drawRight()

```
def drawRight(x,y,r,n):
    x1 = x
    y1 = y
    for i in range(0,n):
        hexagon = mpatches.RegularPolygon(
            (x1, y1),
            6,
            radius=r,
            orientation=np.pi/6,
            facecolor='#b0497e',
            edgecolor='black',
            linewidth=1
        )

        ax.add_patch(hexagon)

        x1 = x1 + 3*r/2
```

drawLeft()

```
def drawLeft(x,y,r,n):
    x1 = x
    y1 = y
    for i in range(0,n):
        hexagon = mpatches.RegularPolygon(
            (x1, y1),
            6,
            radius=r,
            orientation=np.pi/6,
            facecolor='#b0497e',
            edgecolor='black',
            linewidth=1
        )

        ax.add_patch(hexagon)

        x1 = x1 - 3*r/2
        y1 = y1 + np.sqrt(3)*r/2
```

drawTopRight()

```
def drawTopRight(x,y,r,n):
    while n > 0:
        drawRight(x,y,r,n)
        y = y + np.sqrt(3)*r
        n = n -1
```

drawTopLeft()

```
def drawTopLeft(x,y,r,n):
    y1 = y
    while n > 0:
        drawLeft(x,y1,r,n)
        y1 = y1 + np.sqrt(3)*r
        n = n -1
```

drawTop()

```
def drawTop(x,y,r,n):
    k = n
    drawTopLeft(x,y,r,k)
    drawTopRight(x,y,r,n)
```

drawBottomRight()

```
def drawBottomRight(x,y,r,n):
    k = n
    drawTopLeft(x,y,r,k)
    drawTopRight(x,y,r,n)
```

drawBottomLeft()

```
def drawBottomLeft(x,y,r,n):
    x1 = x - 3*r/2
    y1 = y - np.sqrt(3)*r/2
    k = n - 1
    while k > 0:
        drawLeft(x1,y1,r,n-1)
        y1 = y1 - np.sqrt(3)*r
        k = k - 1
```

drawBottom()

```
def drawBottom(x,y,r,n):
    drawBottomRight(x,y,radius,n)
    drawBottomLeft(x,y,radius,n)
```

draw()

```
def draw(x,y,radius,n):
    drawTop(x,y,radius,n)
    drawBottom(x,y,radius,n)
```

A.2 color.py

```
def color(x,y,radius,i,j,n):
    x0, y0 = x1, y1
    distance = np.sqrt(3*(i**2 + i*j + j**2))*radius
    hexagon = mpatches.RegularPolygon((x0, y0), 6,
                radius=radius, orientation=np.pi/6,
                facecolor='#ffaadf', linewidth=1
    )
    ax.add_patch(hexagon)

    x1 = (3+(i-2)*3/2)*radius + x0
    y1 = (j + i/2)*np.sqrt(3)*radius + y0
    for k in range(0,3):
        for p in range(0,2):
            hexagon = mpatches.RegularPolygon(
                        (2*p*x0 + ((-1)**p)*x1, 2*p*y0 +
                        ((-1)**p)*y1),
                        6,
                        radius=radius, orientation=np.pi/6,
                        facecolor='#3c001b', linewidth=1
            )
            ax.add_patch(hexagon)


        a = x1
        x1 = x1 - 2*distance*np.sin(np.pi/6)*np.cos(-np.pi/6 +
        np.arcsin((x1-x0)/distance))
        y1 = y1 + 2*factor*np.sin(np.pi/6)*np.sin(-np.pi/6 +
        np.arcsin((a-x0)/distance))



    ax.set_xlim(-10.5*radius, 10.5*radius)
    ax.set_ylim(-10.5*radius, 10.5*radius)
    plt.show()
```

A.3 api.py

```
def api(x,y,radius,i,j,n):
    draw(x,y,radius,n)
    color(x,y,radius,i,j,n)
```