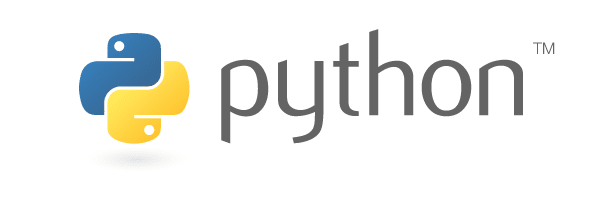


**Rapport de Projet**



**Simulation Python Matplotlib**

Remerciements

Je tiens tout particulièrement à remercier mon encadrant Yulin ZHANG de m’avoir proposé ce sujet pour enrichir mes connaissances en langage de programmation et de m’avoir accordé du temps pour répondre aux questions nécessaires à la réussite de ce projet.

Table des matières

**Table des matières**………………....……………..……..……………..........…3

**Table des figures**……………………………………………………..….….….4

[I. **Introduction** 5](#_Toc48828436)

[a) Déroulement 5](#_Toc48828437)

[b) Diagramme Gantt 5](#_Toc48828438)

[II. **Recherches** 6](#_Toc48828439)

[1) Sujet : 6](#_Toc48828440)

[2) Entrainement : 6](#_Toc48828441)

[III. **Programmation trajectoire** 7](#_Toc48828442)

[1) Numpy et Matplotlib 7](#_Toc48828443)

[2) Classes et fonctions 8](#_Toc48828444)

[3) Trajectoire avec obstacles 8](#_Toc48828445)

[IV. **Détection d’obstacles** 10](#_Toc48828446)

[1) 1er essai 10](#_Toc48828447)

[2) 2ème essai 12](#_Toc48828448)

[3) Assemblage des programmes 13](#_Toc48828449)

[4) Eviter les obstacles 14](#_Toc48828450)

[V. **Programme final** 15](#_Toc48828451)

[1) Programme principal 15](#_Toc48828452)

[2) Initialiser la matrice du plateau 16](#_Toc48828453)

[3) Dessiner plateau 17](#_Toc48828454)

[4) Bouger Turtle 18](#_Toc48828455)

[5) Rendu graphique final 20](#_Toc48828456)

[VI. **Problèmes et Améliorations** 20](#_Toc48828457)

[VII. **Conclusion** 21](#_Toc48828458)

[VIII. **Annexes** 22](#_Toc48828459)

Table des figures

[Figure 1 diagramme Gantt 5](#_Toc48828460)

[Figure 2 exemple sur Matplotlib 6](#_Toc48828461)

[Figure 3 objectif sous python 7](#_Toc48828462)

[Figure 4 code début trajectoire 7](#_Toc48828463)

[Figure 5 exécution n°1 sous Vscode 8](#_Toc48828464)

[Figure 6 module animation 8](#_Toc48828465)

[Figure 7 format des courbes 9](#_Toc48828466)

[Figure 8 création des obstacles 9](#_Toc48828467)

[Figure 9 exécution 10](#_Toc48828468)

[Figure 10 exécution n°2 sous Vscode 10](#_Toc48828469)

[Figure 11 liste taille grille 11](#_Toc48828470)

[Figure 12 fonction color map 11](#_Toc48828471)

[Figure 13 grille avec obstacle 11](#_Toc48828472)

[Figure 14 code matrice + obstacles 12](#_Toc48828473)

[Figure 15 matrice d'obstacles 12](#_Toc48828474)

[Figure 16 exécution matrice obstacles 12](#_Toc48828475)

[Figure 17 matrice aléatoire 13](#_Toc48828476)

[Figure 18 grilles 13](#_Toc48828477)

[Figure 19 code assemblage 13](#_Toc48828478)

[Figure 20 simulation graphique obstacle 14](#_Toc48828479)

[Figure 21 lancement du code 15](#_Toc48828480)

[Figure 22 taille matrice 15](#_Toc48828481)

[Figure 23 exécution des fonctions 15](#_Toc48828482)

[Figure 24 fonction initialisation 16](#_Toc48828483)

[Figure 25 coordonnées obstacles 16](#_Toc48828484)

[Figure 26 obstacle dans matrice 16](#_Toc48828485)

[Figure 27 fonction dessiner le plateau 17](#_Toc48828486)

[Figure 28 caractéristiques du plateau 17](#_Toc48828487)

[Figure 29 bouger turtle 18](#_Toc48828488)

[Figure 30 paramètres de turtle 18](#_Toc48828489)

[Figure 31 déplacement du robot turtle 18](#_Toc48828490)

[Figure 32 déplacement du robot vers le haut 19](#_Toc48828491)

[Figure 33 déplacement du robot vers la droite 19](#_Toc48828492)

[Figure 34 exécution du résultat dans le terminal 19](#_Toc48828493)

[Figure 35 exécution du programme final 20](#_Toc48828494)

## Introduction

Dans le cadre de ma première année de Master en Industrie Numérique à l’INSSET de Saint-Quentin, nous devions effectuer un stage de fin d’année, mais hélas la situation épidémique en France a compliqué les recherches et les acceptations de stage se sont faites rares, ce qui fut amplifié avec le confinement. Pour pallier à ce problème, j’ai effectué un projet de programmation afin d’améliorer mes connaissances en langage Python. En effet, de nature type « mécanicien », j’étais encore novice en programmation avec un niveau assez faible cette année, j’ai donc demandé un projet sur Python afin de relever mon niveau sur ce logiciel et donc espéré avoir un niveau en programmation égal à mon niveau de mécanique. Cela me permettra d’être complet par la suite et d’être polyvalent sur un projet.

### Déroulement

Ce projet fut d’une durée totale d’environ dix semaines, qui s’est étendu du 12 Juin au 20 Août. Durant cette période, on n’était pas en total liberté, il fallait chaque semaine envoyer un rapport détaillé sur nos activités et travaux effectués lors de la semaine. Chaque semaine, nous avions également une revue de projet en ligne pour effectuer une petite présentation de notre avancement et poser des questions sur le projet si nécessaire. Toutes nos données (rapport, programme, présentation) étaient partagées en ligne sur « GitHub » et donc accessible à n’importe quel moment par mon encadrant.

### Diagramme Gantt

Le diagramme de GANTT est un outil efficace de gestion de tâches qui offre une vue globale des tâches à réaliser. Il m’a permis de me fixer une ligne directrice et d’essayer de se projeter dans le futur. Je l’ai modifié au fur et à mesure de l’avancement et des modifications du projet.

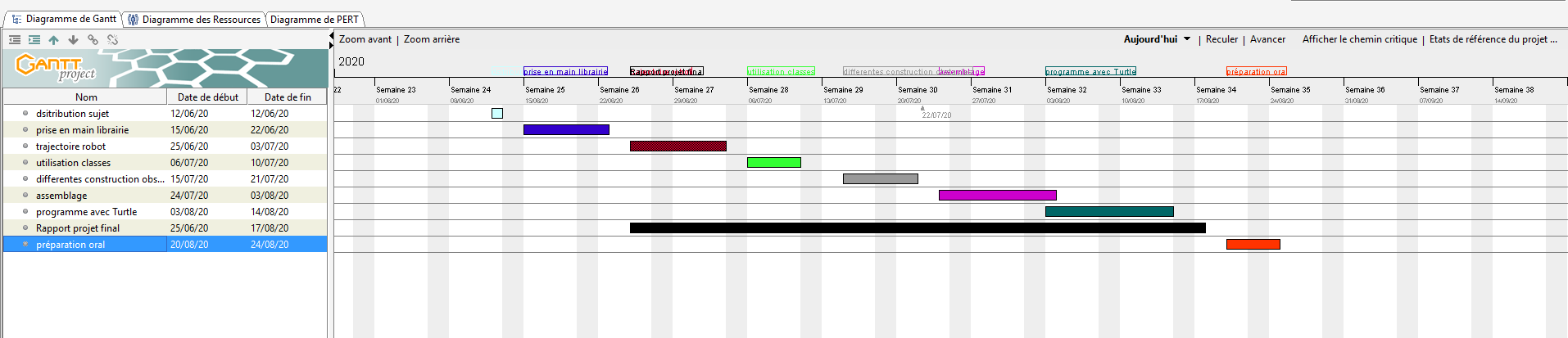


Figure 1 diagramme Gantt

## Recherches

### Sujet :

L’intitulé du sujet a été le suivant :

* Construire une map avec les obstacles en utilisant « Matplotlib »
* Robot doit traverser la map en diagonal en évitant les obstacles

Dans un premier temps, comme nous allons utiliser une bibliothèque du nom de « Matplotlib », j’ai fait des recherches et tenté de m’imprégner de ses fonctions. Pour cela, j’ai imprimé la fiche des fonctions de ce module qu’on peut utiliser à tout moment lors de la création d’un code.

### Entrainement :

J’ai ensuite fait de nombreux exercices débutant avec « Matplotlib » afin de comprendre à maitriser cet outil. Par exemple, je me suis exercé à créer différents types de courbes et changer leurs paramètres (comme une courbe sinusoïdale).

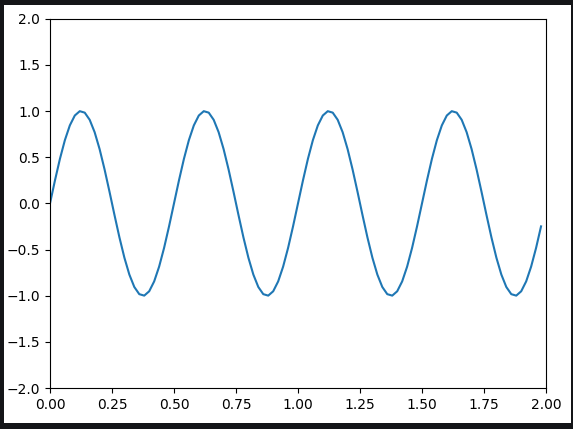


Figure 2 exemple sur Matplotlib

*On retrouve le programme complet en Annexe 1*

Une fois les fonctions de bases acquises, j’ai pu me pencher plus sérieusement sur le projet et son intitulé.

Je me suis donc permis de questionner un peu plus mon encadrant de projet sur l’intitulé dans le but de cerner au mieux le problème et de trouver un code approuvé afin de ne pas faire de hors sujet.

Le but étant de tracer une droite en temps réel et donc non fixe, celle-ci devra éviter des formes géométriques présentes sur une grille.

Cela se traduit par un robot traversant une pièce en diagonal et devant éviter les obstacles afin d’arriver au point d’arrivée.

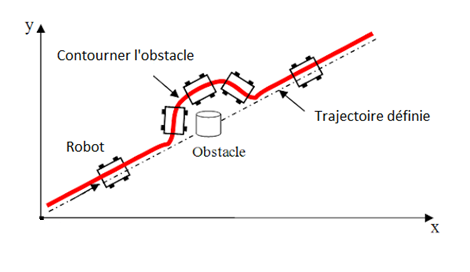


Figure 3 objectif sous python

## Programmation trajectoire

### Numpy et Matplotlib

Une fois la problématique bien définie, j’ai pu commencer la programmation. J’ai d’abord décidé de créer un code contenant :

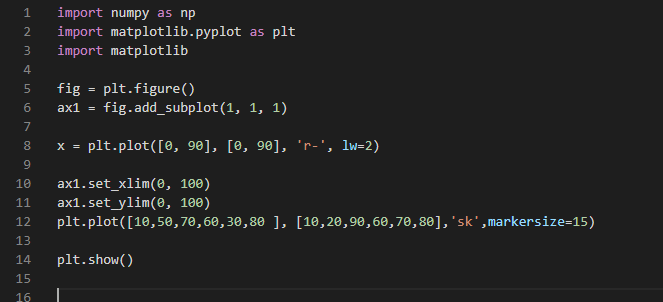


Figure 4 code début trajectoire

1. On importe les librairies que l’on souhaite utilisées (Numpy et Matplotlib)
2. On réalise une figure (une fenêtre graphique vide)
3. On ajoute un graphique gradué de 0 à 100 de taille ‘’1,1,1’’ qui s’ajoute à la figure vide réalisée précédemment
4. On crée des carrés noirs de taille ‘’15’’ qui contiennent certaines coordonnées x, y

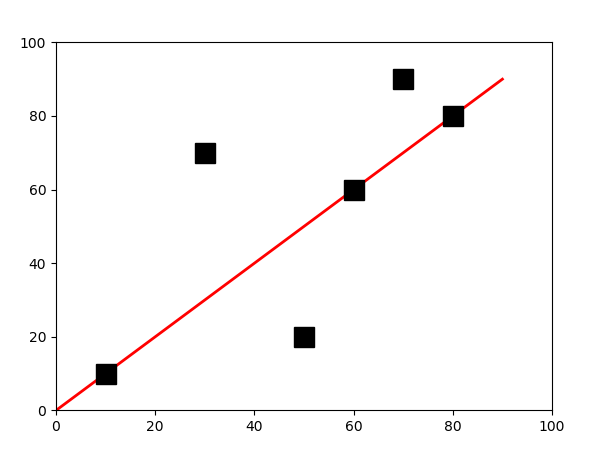


Figure 5 exécution n°1 sous Vscode

Mon prochain but a été d’améliorer le code précédent, avec trois objectifs :

* Compléter avec une droite qui se trace au cours du temps.
* Rendre visible la position du robot a l’instant t virtuellement simulé au bout de cette droite (on crée un marqueur).
* Rendre plus lisible et professionnel le code grâce aux classes et fonctions.

### Classes et fonctions

Qu’est-ce qu’une classe sous python ?

Une classe est un ensemble incluant des variables et des fonctions. Les attributs sont des variables accessibles depuis toute méthode de la classe où elles sont définies. Les classes sont de types modifiables. Elles permettent de rendre plus lisible le code et d’automatiser certaines tâches (ex : classe d’un être humain, on pourra créer différents types d’humain juste en lui indiquant certains paramètres).

### Trajectoire avec obstacles

Pour arriver à tracer la trajectoire en temps réel, j’ai, dans un premier temps, utilisé le module (animation) de Python. Ce module nous permet sous Matplotlib d’animer n’importe quelle courbe en fonction du temps.



Figure 6 module animation

Pour faire fonctionner ce module, j’ai créé une fonction constructeur « \_\_init\_\_ » qui regroupe tous les modules du code précédent et servant d’initialisation :

* Animation.TimedAnimation= nom du module appelé
* *Self :* convention utilisée en Python lors de création d’une classe et d’une fonction, il renvoie à l’objet de la classe créée, donc si on ajoute une variable dans notre nouvelle classe, il portera le nom de « self. variable ».
* *fig*: est la figure dans laquelle l'animation va être tracée.
* *blit=True* : permet de ne pas tout recalculer à chaque fois
* *interval* : correspond à l'intervalle de temps entre 2 images

Enfin, j’ai réalisé, au bout de la courbe qui représente la trajectoire, un petit marqueur (une flèche) qui représentera notre robot à l’instant t.

**Fonctionnement et explications du nouveau code :**

On crée une classe que l’on nomme « SubplotAnimation(animation. TimedAnimation) » qui regroupe les trois fonctions que l’on va utiliser :

* 1ère fonction : Initialisation

-On crée une figure

-On choisit l’échelle (une échelle de 1)

* On trace une courbe de coordonnées x, y d’origine 0, vitesse 20 et précision 256.
* On modifie l’aspect de la courbe (couleur, forme…)
* On réalise une forme que l’on va ajouter au bout de la courbe (ici une flèche pour simuler la position à l’instant t du robot).

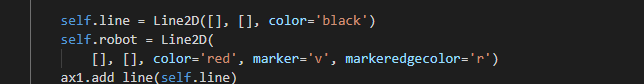


Figure 7 format des courbes

* On ajoute la courbe et la flèche sur la figure instaurée précédemment
* On définit les axes (ici les graduations représentent la taille de la map que l’on souhaite construire)
* On ajoute des formes sur le graphique (qui vont simuler les objets que le robot devra éviter). Ces obstacles ont des formes et des couleurs différentes (noir, bleu, jaune, carré, rectangle).

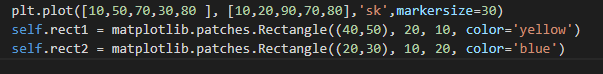


Figure 8 création des obstacles

* On utilise la fonction animation pour lancer le mouvement du robot
* Fonction numéro 2 : qui sert à tracer la courbe et son marqueur avec l’animation en temps réel.

Il faut que la flèche s’efface au fur et à mesure du mouvement car se sont en réalité une multitude de flèches qui s’ajoutent au cours du temps. Pour cela, on crée une nouvelle fonction avec une variable i, celle-ci va s’incrémenter et par la suite va se soustraire afin d’effacer les flèches du positionnement précédent et afin d’obtenir un résultat propre.

On affiche le résultat avec « plt.show() »

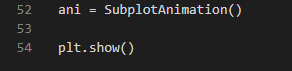


Figure 9 exécution

On obtient la fenêtre graphique suivante :

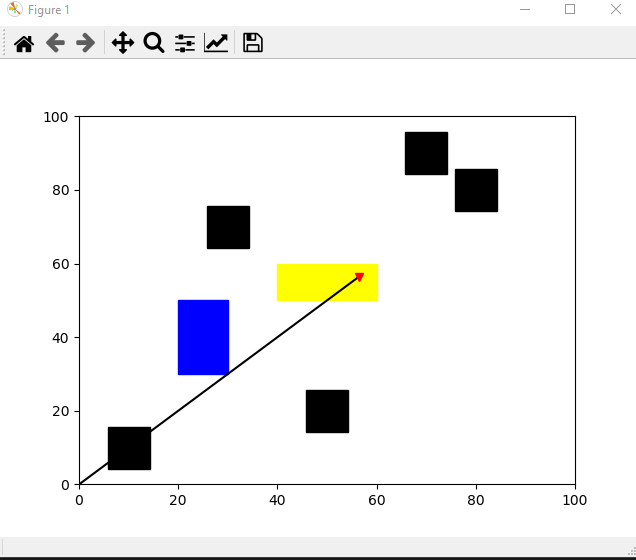


Figure 10 exécution n°2 sous Vscode

*On retrouve le programme complet en Annexe 2*

## Détection d’obstacles

### 1er essai

Maintenant que l’on possède une approche solide du projet et un avancement, notre prochaine étape sera de se consacrer à la détection des obstacles.

Pour cela, j’ai pensé à modifier mes obstacles précédents.

Ainsi, j’ai songé qu’en créant une grille avec certaines cases colorées et en incluant des listes, j’arriverai plus facilement à programmer mon détecteur d’obstacle. Je me suis donc lancé dans cette direction.

Grâce à la grille, on va pouvoir « colorier » des cases qui représenteront des obstacles et que le robot devra éviter par la suite. On souhaite au final avoir une grille de dimension 100x100. Pour commencer, on réalise une grille 10x10 afin d’essayer le bon fonctionnement du programme.

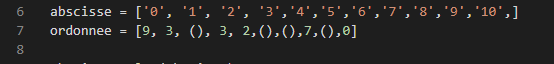


Figure 11 liste taille grille

On crée deux listes « abscisse » et « ordonnée » contenant 10 points avec leur coordonnées x,y. On parcourt ensuite ces deux listes grâce au module ‘len‘.

On utilise la fonction ‘len’ (longueur) qui nous permet d’obtenir la longueur de la chaine du tableau et de l’insérer sur notre figure (graphique) grâce à la fonction mlp.suplots().



Figure 12 fonction color map

Il ne nous reste plus que la mise en forme des obstacles à réaliser. Pour accomplir cela, on utilise la fonction « colormap » qui permet d’obtenir précisément une couleur (rouge ici) grâce aux valeurs définies.

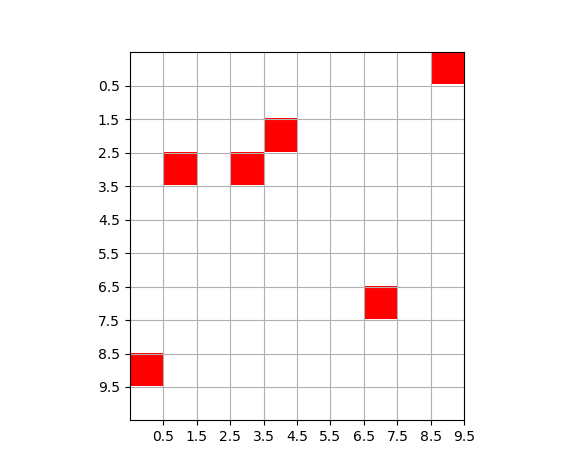


Figure 13 grille avec obstacle

*On retrouve le programme complet en Annexe 3*

### 2ème essai

Finalement, j’ai abandonné cette idée puisque pour faire reconnaitre les cases colorées comme obstacles et les dévier, il faut créer des boucles de difficultés avancées que je ne maîtrise pas. J’ai donc recherché une nouvelle idée.

Le choix le plus judicieux est d’utiliser des matrices remplies de 0 et de 1 :

0 : contient du vide

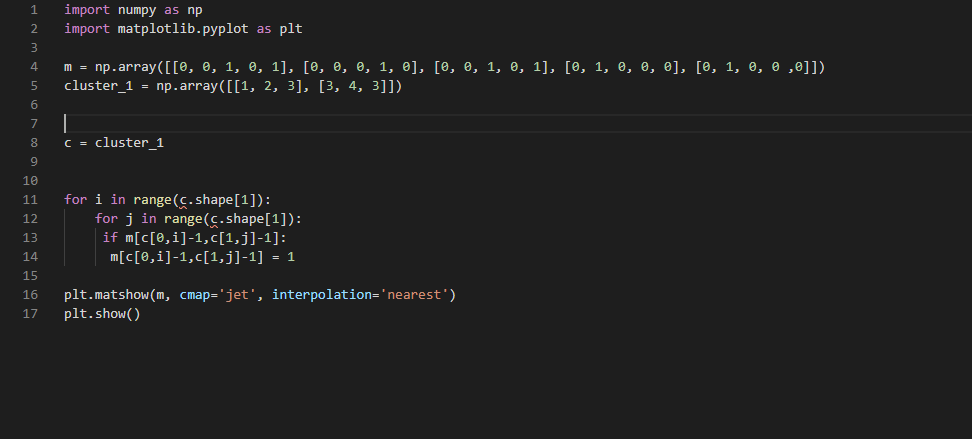
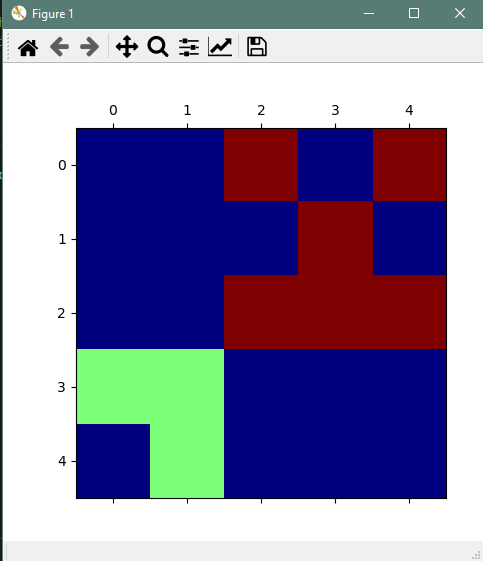
1 : contient un obstacle

Figure 14 code matrice + obstacles

Figure 15 matrice d'obstacles

On rentre manuellement cette matrice grâce à la librairie « numpy » et la fonction array. Par la suite, on va parcourir cette matrice dans une condition « for ». On lui ajoute quelques détails graphiques tels que la couleur avec la fonction colormap.

On obtient ceci :

Ici les obstacles sont définis par l’utilisateur et ne sont donc pas « réaliste ».

Figure 16 exécution matrice obstacles

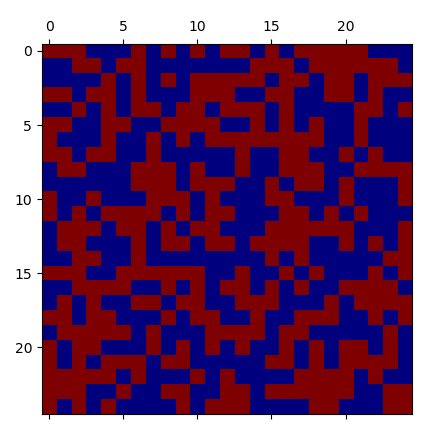
Maintenant que nous avons inséré les obstacles, le prochain objectif est de les générer aléatoirement, et pas dans une matrice définie à l’avance. Pour cela, on crée une matrice aléatoire de 100x100.



Figure 17 matrice aléatoire

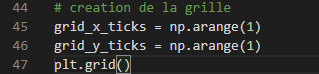
 On crée une grille sur la fenêtre graphique de Python :

Figure 18 grilles

### Assemblage des programmes

### 

Pour le moment, nous avons deux programmes séparés, une trajectoire robot + grille (*Annexe 2*) et une map d’obstacle (*Annexe 2*), le but étant de les assembler pour qu’ils fonctionnent ensemble dans un même programme.

Pour accomplir cela, il faut d’abord revoir la structure des deux programmes. En effet, le programme « trajectoire » est réalisé avec des classes, alors que le deuxième est réalisé sans. Cela peut engendrer des erreurs par la suite.

Il faut refaire les programmes de manière à ce que :

-le 1er programme contienne la map obstacle, grille…

-le 2ème programme contienne le robot et son déplacement.

-le 3ème programme importe les deux précédents (on utilise la fonction « import »)

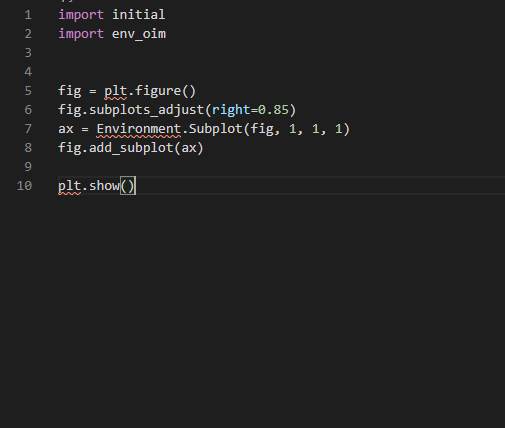


Figure 19 code assemblage

Toutefois, en procédant de cette manière, on constate un **problème** :

Le programme exécute deux fenêtres d’exécutions différentes, comme s’il y avait deux programmes lancés simultanément .

Ce n’est pas ce qu’on recherche puisque l’on souhaite que la « trajectoire » et la « map » soit superposés et réunis dans une seule fenêtre d’exécution.

Pour pallier à ce problème, j’ai assemblé la map et la trajectoire dans un seul « programme » sans les classes, juste pour s’assurer du bon fonctionnement du programme.

### Eviter les obstacles

Je me suis ensuite intéressé à la manière d’éviter les obstacles.

Comme on peut le voir sur l’image ci-dessous, il faut que la trajectoire évite les cases colorées (qui représentent des obstacles), les flèches représentant la trajectoire.

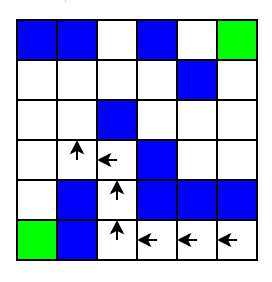


Figure 20 simulation graphique obstacle

Pour réussir cette tâche ardue, j’ai décidé d’abandonner mon programme actuel malgré son avancement et de partir sur un nouveau programme mais avec des librairies différentes.

En effet, je pense plus judicieux et facile d’accomplir ce projet grâce au module « Turtle » déjà inclus dans le logiciel Python.

Le module « Turtle » a une utilisation plus simple que Matplotlib et sa prise en main est assez aisée.

## Programme final

### Programme principal

Le programme principal sert à initialiser certaines choses comme :

* Créer un turtle, spécifier sa forme, sa vitesse d’exécution
* Ici on règle la taille à 400,400

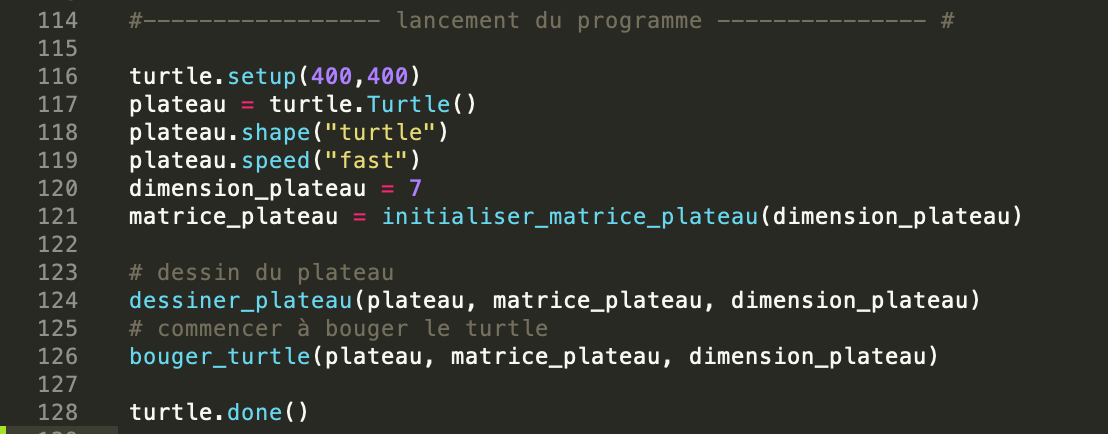


Figure 21 lancement du code

* Déclarer et initialiser la matrice du plateau ainsi que sa dimension (à 7x7).



Figure 22 taille matrice

* Dessiner le plateau avec ses obstacles via la fonction « dessiner\_plateau »



* Faire bouger le turtle dans le plateau

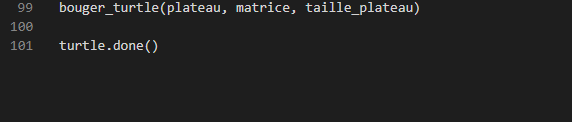


Figure 23 exécution des fonctions

Ceci servait d’initialisation du code, on détaillera un peu plus par la suite les fonctions que ce programme contient afin qu’il évite les obstacles qu’on lui a indiqué.

### Initialiser la matrice du plateau

On crée une première fonction d’initialisation de la map :

* Créer une matrice vide
* Ajouter des lignes à la matrice, le nombre de lignes correspond à la valeur ‘taille\_plateau’
* Chaque ligne va contenir une liste de ‘taille\_plateau’ élément
* Cela crée une matrice de taille (‘taille\_plateau x taille\_plateau’)

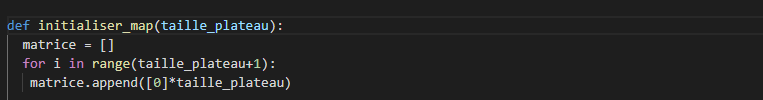


Figure 24 fonction initialisation

* On crée une liste qui va contenir les cordonnées (x,y) des obstacles ( ex : [2,1] est l'obstacle se situant à la colonne 2, ligne 1)
* On peut créer autant d’obstacles que l’on souhaite



Figure 25 coordonnées obstacles

* On génère les obstacles avec la matrice créée précédemment, on définit chaque obstacle dans une position (i, j) dans la matrice
* Ensuite, on indique -1 dans la position de la matrice du plateau
* Puis on retourne à la matrice résultat

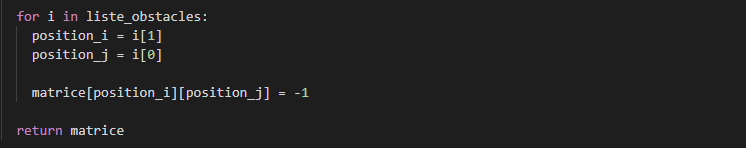


Figure 26 obstacle dans matrice

### Dessiner plateau

Cette fonction sert à :

* Dessiner la liste des obstacles dans le plateau donné
* On parcourt la matrice à 2 indices (lignes et colonnes)
* Si on retrouve une valeur -1 (on a indiqué cela dans la fonction précédente), cela nous indique un obstacle et on dessine un rectangle avec ses cordonnées et ses paramètres correspondants.

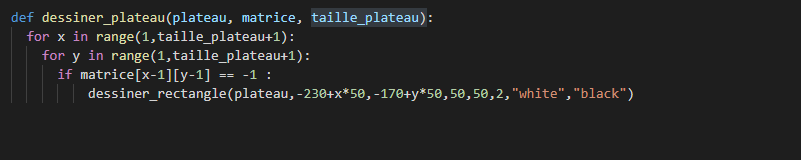


Figure 27 fonction dessiner le plateau

La fonction dessiner rectangle permet de dessiner un rectangle dont les propriétés sont spécifiées aux paramètres :

* Le plateau :'plateau'
* La position du rectangle : 'x' et 'y'
* La taille du rectangle : 'width' et 'height’ (sa ‘largeur’ et ‘hauteur’)
* La taille du stylo : 'size'
* La couleur du le stylo : 'color'
* La couleur de remplissage : 'fill'

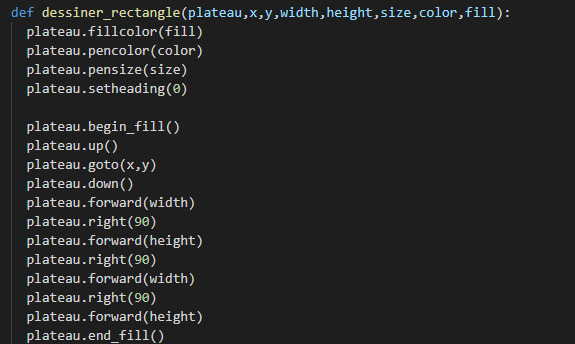


Figure 28 caractéristiques du plateau

### Bouger Turtle

La fonction sert à déplacer le robot dans une direction diagonale de haut à droite, car notre robot n’a pas un déplacement aléatoire mais il avance en diagonal donc du bas à gauche vers le haut droit de la figure. Pour cela, la fonction :

* Fait dessiner un carré rouge plein au point de départ choisit
* Fait changer la vitesse à la plus lente, afin de visualiser le mouvement de notre robot
* Place le robot turtle en bas à gauche du plateau.

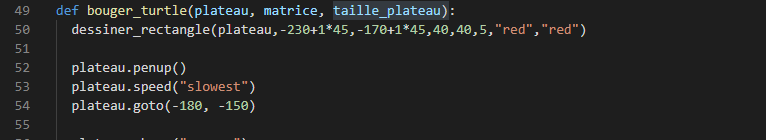


Figure 29 bouger turtle

* Forme le robot comme carré rouge
* Change son apparence classique de « grosse turtle rouge » en un petit carré rouge
* On active le stylo pour avoir un suivi de la trajectoire du robot

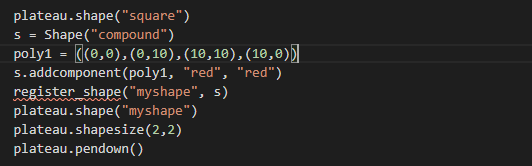


Figure 30 paramètres de turtle

* Commencer le déplacement du robot en partant de la position en bas à gauche
* Dans la matrice cela va à 0,0 (on met 1,1) pour simuler le design
* On vérifie si le robot est bloqué

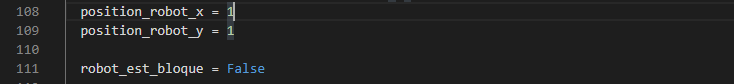


Figure 31 déplacement du robot turtle

On bouge le robot « tant qu’il » n’est pas bloqué par plusieurs obstacles autour de lui d’une manière alternative soit ***vers le haut*** :

* Si le robot est bloqué on passe en « True »
* On va donc vérifier s’il y a un obstacle en haut du robot, s’il n’y a pas d’obstacle on passe en « False » et le robot avance.
* On ajoute +1 à la coordonnée y du robot

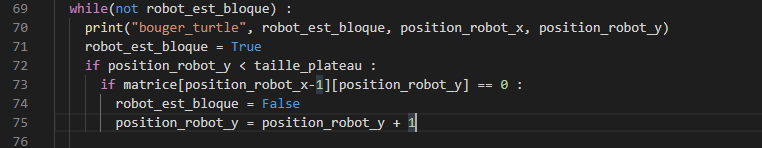


Figure 32 déplacement du robot vers le haut

***Soit vers la droite :***

Si le robot est bloqué par le haut, il va tenter d’aller à droite, même principe que les conditions du dessus sauf qu’on change de coordonnées, on passe en « x ».

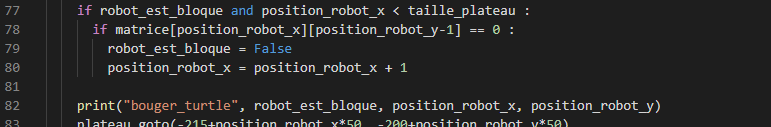


Figure 33 déplacement du robot vers la droite

On a mis en priorité le « if » qui va vers le haut donc cela veut dire que si le robot a le choix entre aller à droite et en haut (imaginons qu’il n’y a pas d’obstacles sur ces deux cases) il ira forcément en haut en priorité et pas à droite.

On y inclut deux « prints » pour avoir un suivi sur la trajectoire du robot empruntée qui s’affiche dans le terminal, et donc voir s’il y a un obstacle qui pose problème.

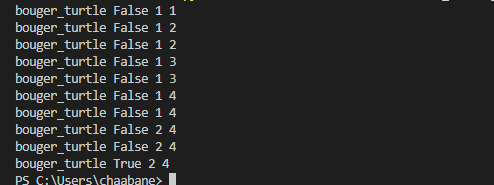


Figure 34 exécution du résultat dans le terminal

*On retrouve le programme complet en Annexe 4*

### Rendu graphique final

Aperçu de l’exécution finale sur ***Vscode*** on obtient ceci***:***

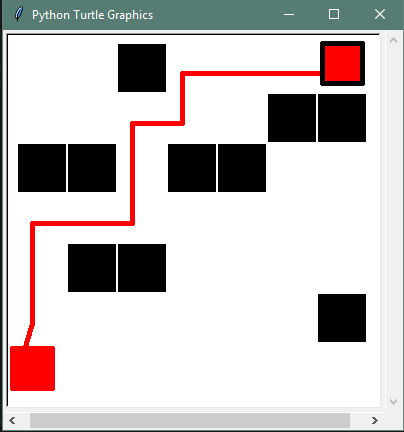


Figure 35 exécution du programme final

## Problèmes et Améliorations

Le plus gros problème de ce code est que si le robot est bloqué par deux obstacles (un à droite et l’autre en haut), il sera bloqué car il n’a pas de fonction pour retourner en arrière et ainsi trouver un nouveau chemin qui sera sans obstacles.

Pour une simulation cela n’est pas gênant mais si on se base sur un cas réel, on n’imagine mal un robot bloqué au milieu d’une pièce à cause de deux obstacles.

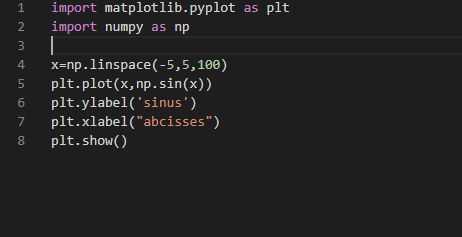
## Conclusion

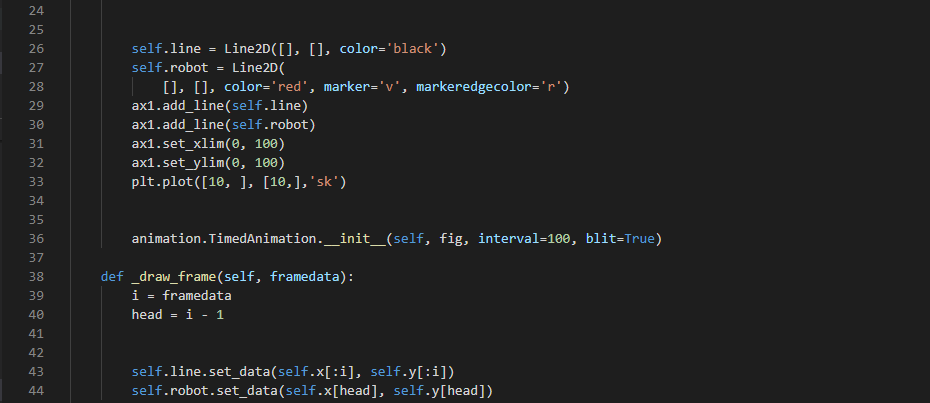
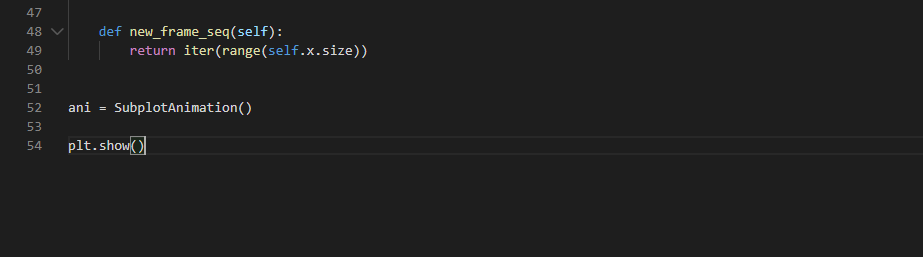
.

La réalisation de ce projet m’a permis de progresser en langage informatique et algorithmique notamment grâce au logiciel Python et ses nombreuses librairies avec lequel je suis désormais plus à l’aise. En effet, j’ai démarré ce projet avec du « Matplotlib », en y incluant du « Numpy » et de l’animation. Puis, je me suis finalement tourné vers du « Turtle ». J’ai donc utilisé un éventail assez large de librairie.

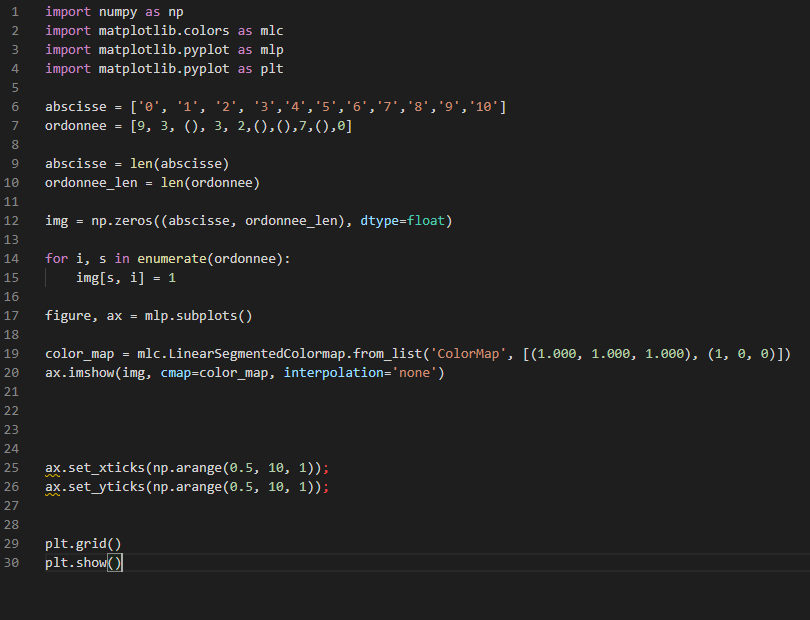
Toutefois, le contexte assez particulier de cette année, m’a poussé à faire davantage de recherche individuelle. J’ai donc dû m’organiser pour trouver les réponses à mes problèmes par moi-même. Ce projet a donc été bénéfique pour moi tant au niveau intellectuel que professionnel et me servira dans ma future vie professionnelle.

## Annexes

Annexe 1 :

Annexe 2 :

Annexe 3 :



Annexe 4 :

import turtle

from turtle import \*

import random

def initialiser\_map(taille\_plateau):

  matrice = []

  for i in range(taille\_plateau+1):

   matrice.append([0]\*taille\_plateau)

  liste\_obstacles = [ [2,1], [2,2],[4,0], [4,1], [4,3], [4,4], [5,6], [1,6],[5,5],[6,2]]

  for i in liste\_obstacles:

    position\_i = i[1]

    position\_j = i[0]

    matrice[position\_i][position\_j] = -1

  return matrice

def dessiner\_rectangle(plateau,x,y,width,height,size,color,fill):

  plateau.fillcolor(fill)

  plateau.pencolor(color)

  plateau.pensize(size)

  plateau.setheading(0)

  plateau.begin\_fill()

  plateau.up()

  plateau.goto(x,y)

  plateau.down()

  plateau.forward(width)

  plateau.right(90)

  plateau.forward(height)

  plateau.right(90)

  plateau.forward(width)

  plateau.right(90)

  plateau.forward(height)

  plateau.end\_fill()

def dessiner\_plateau(plateau, matrice, taille\_plateau):

  for x in range(1,taille\_plateau+1):

    for y in range(1,taille\_plateau+1):

      if matrice[x-1][y-1] == -1 :

          dessiner\_rectangle(plateau,-230+x\*50,-170+y\*50,50,50,2,"white","black")

def bouger\_turtle(plateau, matrice, taille\_plateau):

  dessiner\_rectangle(plateau,-230+1\*45,-170+1\*45,40,40,5,"red","red")

  plateau.penup()

  plateau.speed("slowest")

  plateau.goto(-180, -150)

  plateau.shape("square")

  forme = Shape("compound")

  carre = ((0,0),(0,10),(10,10),(10,0))

  forme.addcomponent(carre, "red", "red")

  register\_shape("myshape", forme)

  plateau.shape("myshape")

  plateau.shapesize(2,2)

  plateau.pendown()

  position\_robot\_x = 1

  position\_robot\_y = 1

  robot\_est\_bloque = False

  while(not robot\_est\_bloque) :

    print("bouger\_turtle", robot\_est\_bloque, position\_robot\_x, position\_robot\_y)

    robot\_est\_bloque = True

    if position\_robot\_y < taille\_plateau :

      if matrice[position\_robot\_x-1][position\_robot\_y] == 0 :

        robot\_est\_bloque = False

        position\_robot\_y = position\_robot\_y + 1

    if robot\_est\_bloque and position\_robot\_x < taille\_plateau :

      if matrice[position\_robot\_x][position\_robot\_y-1] == 0 :

        robot\_est\_bloque = False

        position\_robot\_x = position\_robot\_x + 1

    print("bouger\_turtle", robot\_est\_bloque, position\_robot\_x, position\_robot\_y)

    plateau.goto(-215+position\_robot\_x\*50, -200+position\_robot\_y\*50)

  plateau.speed("fast")

  dessiner\_rectangle(plateau,-230+7\*50+5,-170+7\*50,40,40,5,"black","red")

  plateau.penup()

  plateau.goto(-215+position\_robot\_x\*50, -200+position\_robot\_y\*50)

turtle.setup(400,400)

plateau = turtle.Turtle()

plateau.shape("turtle")

plateau.speed("fast")

taille\_plateau = 7

matrice = initialiser\_map(taille\_plateau)

dessiner\_plateau(plateau, matrice, taille\_plateau)

bouger\_turtle(plateau, matrice, taille\_plateau)

turtle.done()