# Quelques implémentations de la POOP

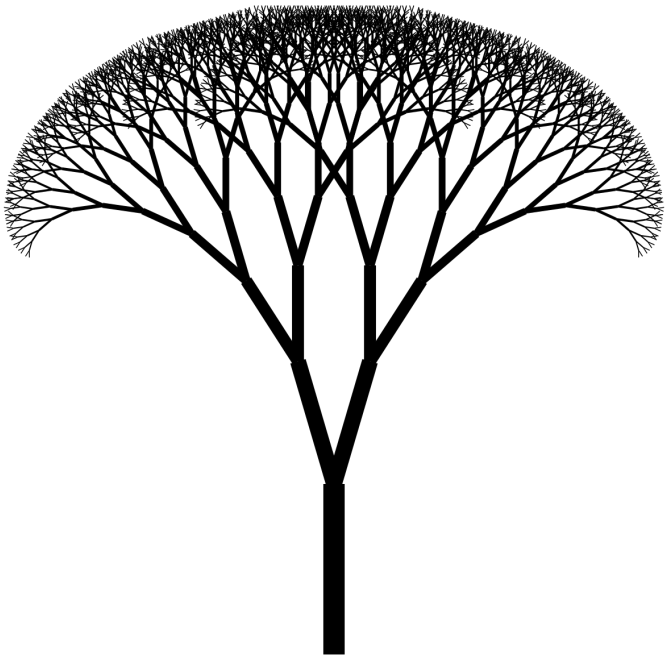
Club Code

September 12, 2021

## Contents

# 1 Les arbres

Dans cette partie nous allons tracer nos premières fractales : des arbres ! Notre objectif est d'obtenir dans un premier temps un arbre similaire à celui ci dessous. Pour ce faire on peut utiliser la pouissance de la récursion.



L'idée est de dessiner l'arbre branche par branche (c'est à dire segment par segment) en imposant à chaque branche une longueur et un angle avec la branche précédente. On part du principe que chaque branche se subdivise en 2 nouvelles branches plus petites qui elles même vont se subdiviser d'où l'utilisation d'une fonction récursive pour le tracé de cet arbre.

Vous pouvez utiliser les bibliothèques graphiques et le langage de programmation de votre choix pour écrire cette fonction. Si vous ne savez pas lesquels choisir nous recommandons le module matplotlib.pyplot en python qui est simple à utiliser.

Si vous avez du mal n'hésitez pas à prendre le temps de faire un dessin sur papier et d'essayer d'executer votre algorithme à la main.

## 2 Creepers... Oh Man...

Dans le jeu vidéo Minecraft - qu'on ne présente plus -, le Creeper est le cauchemar des constructeurs. On ne va pas coder tout le jeu en quelques heures cependant, et certainement pas en Python. Le but ici est de travailler la notion d'héritage, de manière similaire au graphique donné dans le cours.


**Oui, c'est sooo 2012.**

Il s'agit d'abord de créer une classe Entity, qui possède un identifiant, un nombre de point de vie, une valeur d'attaque et des coordonnées spatiales. Une entité peut prendre des dégâts (méthode `hurt`), voire mourir (méthode `die`) si ses points de vie sont inférieurs à 0. Cette méthode indique dans le chat que l'entité est morte. Elle peut également attaquer une autre entité (méthode `attack`), lui infligeant autant de dégât que sa valeur d'attaque. Enfin, une entité peut se déplacer (méthode `move`). Le mouvement est relatif. Une Entity prends en arguments une liste, nommée entity_list, lui permettant d'avoir accés à la liste des entités. Elle s'y rajoute dans sa méthode `__init__`.

Un Player est une Entity qui possède un pseudo, un nombre de point de vie et de point de nourriture initialement à 20, et une arme (vous pouvez créer une classe Weapon si vous voulez, mais on peut également juste lui donner un nom et ne pas aller plus loin). Sa valeur d'attaque est de 2. Un joueur peut utiliser la méthode `chat` pour envoyer un message. Un joueur spawn toujours aux coordonnées (0;0).

Un Creeper est une Entity qui possède un nombre de point de vie de base de 10, qui fait faire son bruit caractéristique ("Hissssss") grâce à la méthode `make_sound`, et qui a une attaque spéciale: en effet, il explose, faisant 8 points de dégâts à toutes les entités autour de lui dans un rayon de 3m, et se tuant au passage. Il faut donc parser la liste des entités lorsqu'il explose.

Un Zombie est une Entity qui possède un nombre de point de vie de base de 15, qui fait faire son bruit caractéristique ("Growl...") grâce à la méthode `make_sound`, et qui attaque en faisant 4 points de dégât. S'il se déplace à moins d'un mètre d'un joueur, il le frappe automatiquement.

Il faudra donc parser la liste des entités pour trouver un joueur.

Pour tester votre code, rajouter les lignes suivantes à votre code:

```
entity_list = []
steve = Player(id=42, pseudo="Steve")
creeper = Creeper(id=69, x=1, y=1)
zombie = Zombie(id=666, x=3, y=2)

steve.chat("Oh boy, I sure do love being alive !")

zombie.move(-2.5, -1.5)
zombie.make_sound()
creeper.make_sound()
creeper.attack()
```

La sortie devrait ressembler à cela:

```
Steve: "Oh boy, I sure do love being alive !"
Growl...
Hisssssss
Creeper has blown up.
Steve has died.
Zombie has died.
```

# 3 Entrer dans la matrice

Vous avez probablement déjà utilisé la bibliothèque `numpy` qui met à disposition de nombreuses classes et méthodes mathématiques. L'idée ici est d'implémenter partiellement le concept de matrice, et ainsi mettre en œuvre des définitions d'opérateurs et des méthodes de classe.

Dans un premier temps, nous avons besoin d'une classe `Matrix`, munie d'un constructeur prenant en paramètre une liste à deux dimensions, et fixant les attributs `matrix` et `shape` (étant un tuple de 2 nombres : lignes puis colonnes).

**Pouet pouet j'ai fait une blague.**

Définissons maintenant quelques opérateurs :

- `__add__(self, m)` : renvoie une matrice dont les termes ont été ajouté un à un. On prendra le soin de vérifier que les dimensions sont compatibles.
- `__sub__(self, m)` : renvoie une matrice dont les termes de la première se font soustraire ceux de la deuxième un à un.
- `__mul__(self, x)` : renvoie la matrice résultante de la multiplication de chaque terme de la matrice par le réel.
- `__rmul__(self, x)` : est égale à `__mul__` et permet la commutativité de cette multiplication par un réel.
- `dot(self, m)` : renvoie la matrice résultante de la multiplication matricielle entre la première et la deuxième matrice. On prendra le soin de vérifier que le nombre de lignes de la première est égal au nombre de colonnes de la seconde. Chaque terme de la matrice résultante s'obtient à partir de cette formule : $c_{i,j} = \sum_{k=1}^{n} a_{i,k} \cdot b_{k,j}$.

On peut ensuite définir quelques méthodes de classe pratiques :

- `fill(cls, x, shape)` : crée une nouvelle matrice de la taille donnée, dont chaque terme est initié à la valeur précisée.
- `zeros(cls, shape)` : un simple raccourci vers fill, avec x fixé à 0.

- `identity(cls, size)` : crée une nouvelle matrice carrée et identitaire de la dimension voulue.

Et pour finir, une petite méthode en bonus :

- `transpose(self)` : renvoie la transposée de la matrice, où chaque terme suit $b_{i,j} = a_{j,i}$.

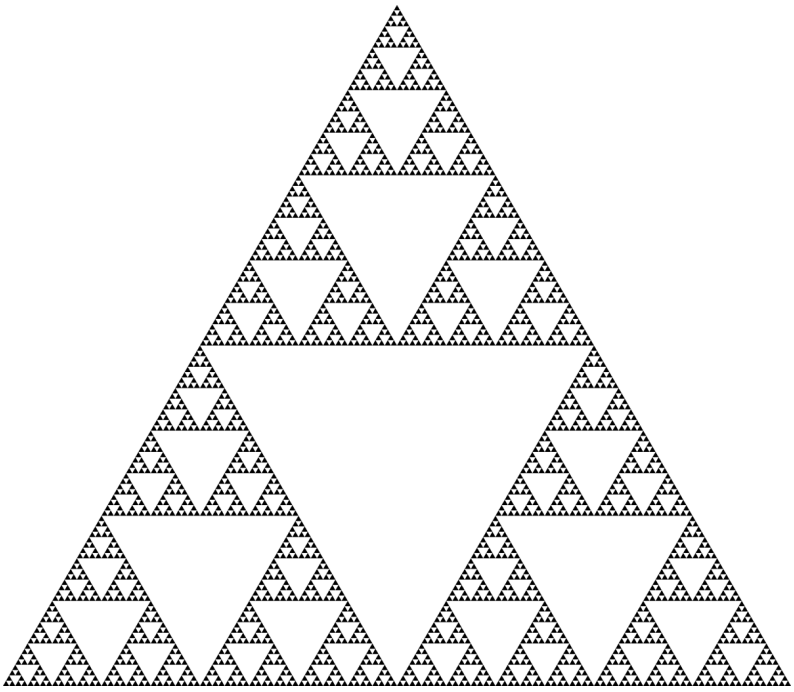
# 4 Bonus : Le triangle de Sierpiński

Il s'agit là d'une fractale classique qui ravira les amateurs des jeux Zelda. On peut retrouver cette fractale dans de nombreux domaine des mathématiques notamment dans le problème classique des tours de Hanoi. Il existe donc de nombreuse façon de la retrouver mais ici nous allons uniquement nous intéresser à un algorithme de dessin intuitif.

Pour la tracer on peut procéder de façon similaire aux arbres en utilisant une fonction récursive. L'idée de cette fonction est de tracer le triangle correspondant aux coordonnées en arguments puis d'appeler récursivement cette fonction sur les 3 sous triangles formés à l'intérieur du triangle en argument.

Comme toujours pensez à faire un dessin pour trouver les coordonnées des points qui forment les sous triangles et ne pas oublier le cas de base sinon votre fonction ne finira jamais de s'exécuter.

Si tout se passe bien vous devriez obtenir un dessin similaire à celui ci.



# 5 Bonus : L'ensemble de Mandelbrot

Warning: Cet ensemble repose sur quelques notions de maths notamment sur les nombres complexes si vous n'êtes pas à l'aise sur ces notions n'hésitez pas à poser des questions !

Pour ce qui est des définitions mathématiques, introduisons tout d'abord la suite $(u_n)_{n\in\mathbb{N}}$ (à valeurs complexes et qui dépend d'un paramètre noté $c$) définie par :

$$\begin{cases} u_0 = 0 \quad et \quad c \in \mathbb{C} \\ \forall n \in \mathbb{N} \quad u_{n+1} = u_n^2 + c \end{cases}$$

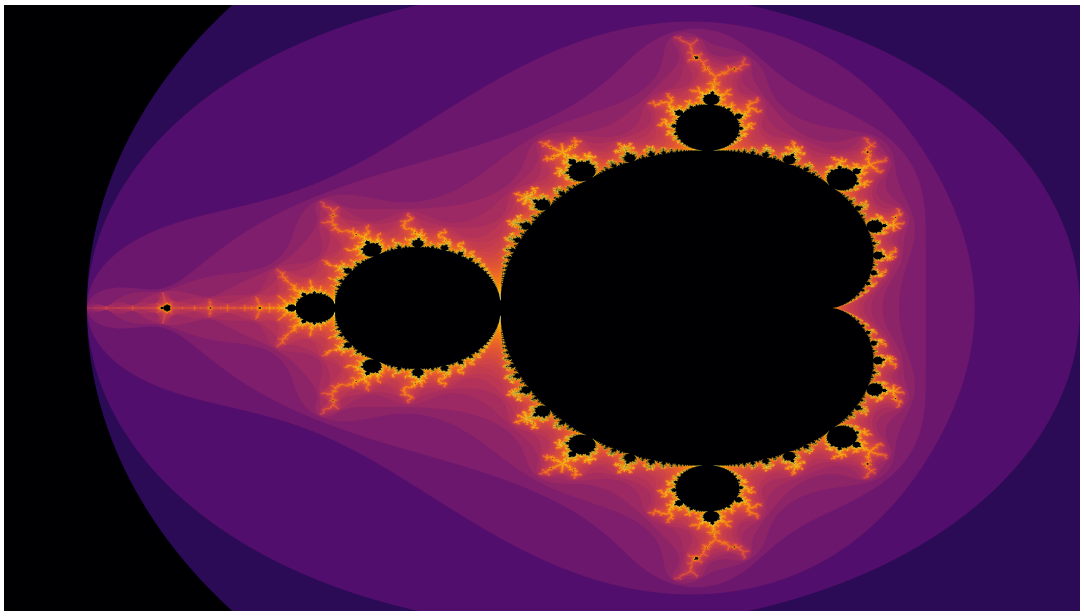
L'ensemble de Mandelbrot est alors défini comme l'ensemble des valeurs de $c$ telles que la suite $(u_n)_{n\in\mathbb{N}}$ ne diverge pas, c'est à dire que $\lim\limits_{n\to+\infty} |u_n| \neq +\infty$

Cette condition est évidemment assez difficile à vérifier si elle est écrite de cette manière. On peut toutefois se convaincre d'un résultat simple qui nous évitera de calculer inutilement les valeurs de la suite : Si à un rang donné $n_0 \in \mathbb{N} \quad |u_{n_0}| \geq 2$ alors la suite diverge[1].

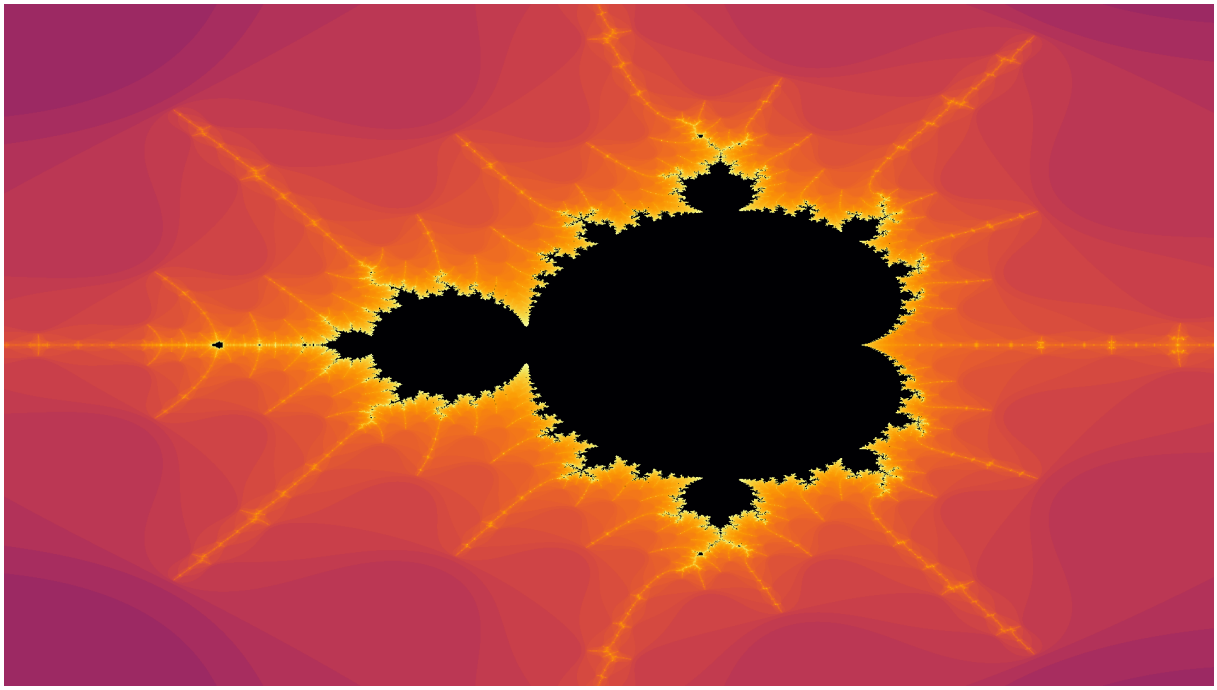
Cette nouvelle condition est bien plus simple à vérifier et c'est elle que nous allons utiliser afin de déterminer si un point appartient oui ou non à l'ensemble de Mandelbrot. Cependant ce n'est pas encore suffisant pour passer à la pratique car on ne peut pas être sûr et certains que pour un $n_0$ très grand le module de $u_{n_0}$ reste en dessous de 2. On va devoir s'arrêter à partir d'un certain nombre de terme de la suite sinon notre programme ne peut pas se terminer. Il faut donc choisir judicieusement (par défaut 100) ce nombre de termes à calculer trop grand notre programme sera très lent, trop petit on perd en précision en incluant à tord des points dans notre ensemble de Mandelbrot.

L'idée est donc de trouver quels points appartiennent à cet ensemble et de les marquer à l'aide d'une petite fonction qui calcule les 100 premiers termes de la suite avec c (argument de cette fonction) comme paramètre et retourne un booléen (ou un entier pour savoir le rang auquel elle diverge ou -1 sinon, ce qui peut être utile pour faire des dégradés de couleur). Ensuite il suffit d'itérer sur un domaine voulu dans $\mathbb{C}$ et de colorier le point correspondant sur une image.

Vous devriez obtenir une image ressemblant à celle ci:



[1]Note du Nouveau Bureau : Verge.



**Images obtenues l'année dernière lorsque j'ai fait ce TP. La seconde est un zoom sur le petit bulbe à gauche ! - Khâtharsis**