Document de rendu de projet - Phase finale

I. Généralités

Tout d'abord, pour mieux appréhender les explications ci-dessous, voici l'architecture du projet :

```
├── index.html
├── assets
│   ├── accueil
│   │   ├── 28.png
│   │   ├── 28compressee.png
│   │   ├── accueil.png
│   │   ├── collatz.jpeg
│   │   ├── valeurs.png
│   ├── tentatives_resolutions
│   │   ├── arbre.jpg
│   │   ├── code.png
│   │   ├── code_opti.png
│   ├── voca
│   │   ├── 12.png
│   │   ├── 20.png
│   │   ├── 27.png
│   │   ├── 30.png
│   ├── icones
│   │   ├── hamburger.png
│   │   ├── icon.png
|   | ├── telecharger.png
├── js
│   ├── simu
|   | ├── fonction.js
|   | ├── app.js
│   ├── responsive.js
│   ├── anim.js
├── css
│   ├── general.css
│   ├── simulateur.css
│   ├── section
│   │   ├── presentation.css
│   │   ├── carte.css
├── src
│   ├── infos.html
│   ├── simulateur.html
│   ├── tentatives-resolutions.html
│   ├── voca.html
```

Voici les éléments redondants sur le site ainsi qu'une description de quelques spécificités techniques qui leur sont propres :

- La navbar : alignée en haut de la page à droite, celle-ci utilise flexbox pour la mise en forme et les noms des pages auxquelles elle renvoie sont dans la police Barlow

- Le footer : prenant toute la largeur de la page et étant situé tout en bas de celle-ci, le footer donne accès à une page de source (infos.html) et à des liens externes (GitHub, Gmail et Drive) dans une police identique à la barre de navigation

- L'aspect responsive : toutes les pages admettent un aspect pouvant s'adapter aux tailles d'écrans plus petites, notamment celle des mobiles. Géré par le fichier responsive.js du dossier js, cet aspect se manifeste notamment par le changement de mise en place du texte et des images dans les sections, passant de côte à côte à côte à l'un de l'autre, mais aussi par la disparition de la barre de navigation au profit d'un menu sandwich pour les tailles d'écrans inférieures ou égales à 633px.

- L'utilisation de la bibliothèque MathJax permettant un rendu LaTex des expressions mathématiques dont les formules sont tapées entre \(\) ou \[\]. Au fur et à mesure de l'élaboration du site, notamment de la page tentatives-resolutions.html, le besoin d'expressions mathématiques propres, claires et agréables à lire s'est de plus en plus fait ressentir, d'où l'utilisation de cet outil. Seul point noir : la couleur des expressions était bien plus foncée que l'écriture du site, ce qui cassait l'identité visuelle d'une page. Pour y remédier, j'ai mis dans le fichier general.css un code permettant de rendre moins foncé le tout : .MathJax.CtxtMenu_Attached_0 { color: rgb(85 85 85); }.

Concernant le choix du découpage du code stylisant la page en 5 fichiers distincts, ce choix est notamment justifié par

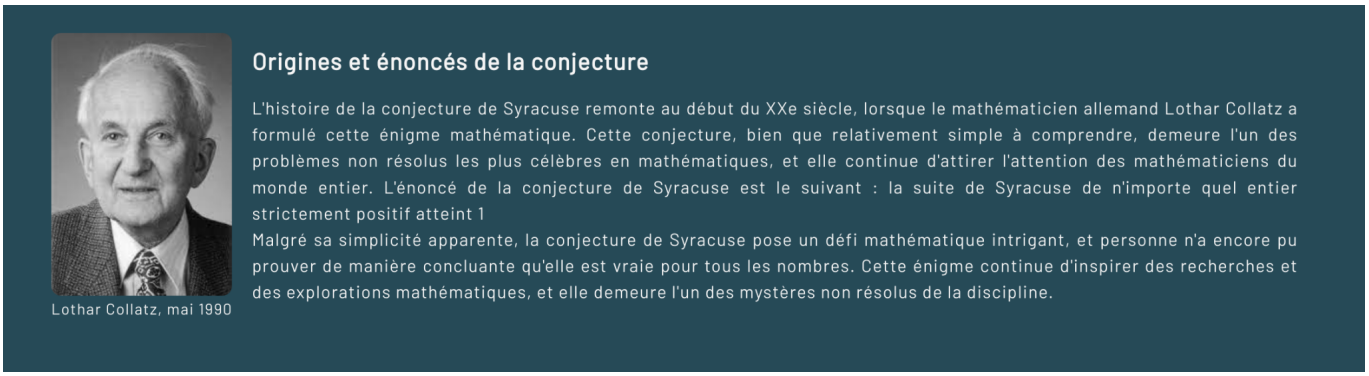
- une meilleure maintenabilité du code (il est plus simple de chercher dans un fichier de ~100 lignes que dans un fichier de plus de 600 lignes)

- de meilleures performances : on importe dans la page uniquement ce dont elle a besoin : il est inutile de charger le style d'une section qui n'y est jamais utilisée

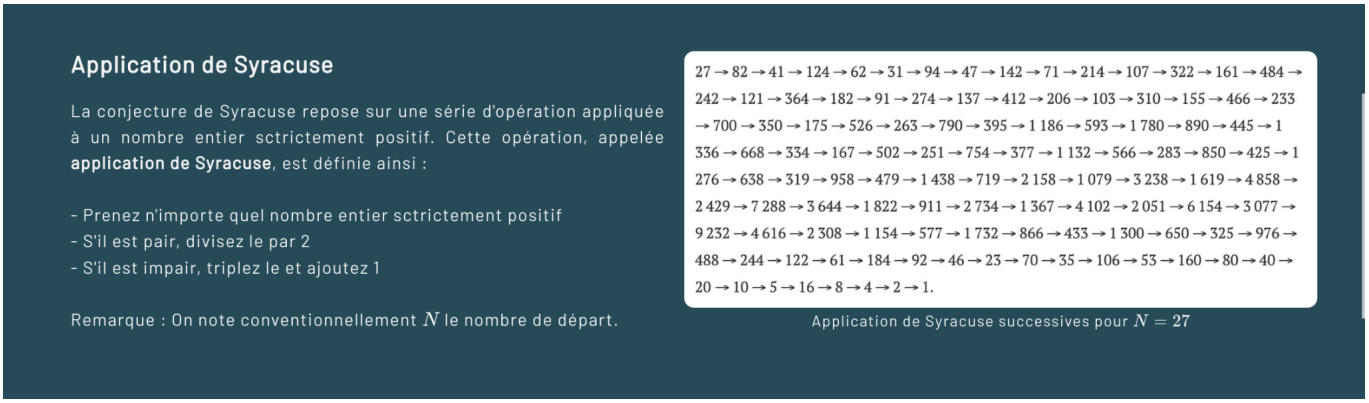
- une plus grande simplicité dans la gestion des polices et de l' écriture. En effet, par exemple, la couleur utilisée pour ajuster la lisibilité des expressions LaTex générées par MathJax change en fonction de la page et du type de section utilisé. Si le fond est bleu comme dans les pages index.html et voca.html, il est plus judicieux que la couleur soit blanche. À contrario, lorsque le fond est blanc, une couleur plutôt grise vers noire sera bien meilleure. Pour gérer ce détail, grâce au découpage des fichiers gérant le style des sections, on importe le code de style d'une seule section dans lequel la couleur est gérée. (si nous importions un seul fichier contenant tout le style relatif aux sections, et bien que cela soit gérable, nous aurions pu avoir des soucis de priorité rendant tantôt le texte blanc tantôt le texte noir vu qu'il est difficile de manipuler la couleur du texte d'une extension qui n'est pas la notre). Ainsi, dans le fichier css se trouve les fichiers :

- general.css contient le code relatif aux éléments redondants cités plus haut (navbar, footer). Il est importé sur toutes les pages du site et contient également le code de style de la page d'accueil car celui-ci est assez court.
- simulateur.css qui possède une page à part en raison de la quantité de code importante (~140 lignes). Le détail du fonctionnement du code est donné plus bas.
- section/presentation.css : ce fichier contient le code relatif au style des sections ayant la classe presentation comme index.html et voca.html. Celles-ci sont les plus utilisées et servent à exposer un concept ou à expliquer une notion nécessitant un exemple visuel comme un graphique. Le style alterne entre image à gauche et texte à droite et texte à gauche image à droite grâce à la propriété flex: 1 appliqué à la div contenant le texte. Cette alternance est gérée par le premier élement présent dans la div, soit la figure soit la div de texte :

Section de classe "presentation" avec la balise figure en premier



Section de classe "presentation" avec la balise div de classe "texte-presentation" en premier



- section.carte.css ; ce fichier contient le code relatif au style des sections ayant la classe carte comme tentatives-resolutions.html et infos.html. Celles-ci sont notamment utilisées dans la page tentatives-resolutions.html et servent plutôt à laisser de l'espace au texte qui peut prendre toute la largeur de l'écran. Celles-ci s'avèrent très utiles lors de la démonstration de notions mathématiques comme l'unicité du cycle de longueur 3

Section de classe "carte" sans image

2. Recherche de cycles non triviaux

Une autre approche pour vaincre la conjecture de Syracuse serait de prouver qu'il existe un cycle différent du cycle trivial.
Pour ce faire, raisonnons par l'absurde et admettons qu'il existe un autre cycle de longueur 3 (A, B, C) tel que $A < B < C$ et

$$A \to B \to C \to A$$

Nous avons donc nécessairement A impair car, le cas échéant, B serait divisé par 2 et $B < A$.
Ainsi $B = 3A + 1$, qui est pair donc $C = \frac{B}{2} = \frac{3A+1}{2}$. Puisque $C \to A$ et $C > A$, C est pair et vaut $\frac{B}{2} = \frac{1}{2}\left(\frac{3A+1}{2}\right) = \frac{3A+1}{4}$
Résolvons maintenant l'équation d'inconnue $A \in \mathbb{N}^*$ afin de déduire les valeurs de A, B, C tels que le cycle existe.

$$A = \frac{3A+1}{4} \iff 4A = 3A + 1 \iff A = 1$$

Ainsi $B = 3 \times 1 + 1 = 4$ et $C = \frac{B}{2} = \frac{4}{2} = 2$
Nous concluons donc que le seul cycle de longueur 3 existant est le cycle trivial.
Par des méthodes de démonstration par l'absurde, il est possible de prouver qu'il n'existe pas de cycle de longueur 2 ou 5.
En général, le théorème de Eliahou nous dit qu'il n'existe aucun cycle de nombre positif inférieur à 180 000 000 000.

Section de classe "carte" avec image illustrative

- Les nombres de la forme $2k$ et 2^n, pour tout $n \in \mathbb{N}^*$, car ceux ci sont directement divisés par 2 donc $u_1 < N$

- Les nombres impairs de la forme $4k + 1$ car,

$$4k + 1 \to 12k + 4 \to 6k + 2 \to 3k + 1$$

et $3k + 1 < 4k + 1 \iff u_3 < N$
Les nombres de la forme $4k + 1$ atterissent donc.

- Les nombres de la forme $16k + 3$ car

$$48k + 10 \to 24k + 5 \to 72k + 16 \to 36k + 8 \to 18k + 4 \to 9k + 2$$

et $9k + 2 < 16k + 3 \iff u_6 < N$
Les nombres de la forme $16k + 3$ atterissent donc.

(Où $\to$ désigne l'application de syracuse)

```python
def suite_de_syracuse(nombre_a_tester: int) -> list[int]:
    """Renvoie la suite de syracuse d'un nombre passé en argument"""
    sequence = [nombre_a_tester]
    while nombre_a_tester != 1:
        if nombre_a_tester % 2 == 0:
            nombre_a_tester //= 2
        else:
            nombre_a_tester = (nombre_a_tester * 3 + 1) // 2
        sequence.append(nombre_a_tester)
    return sequence


1 usage
def est_exclu(nombre_a_tester: int) -> bool:
    """Renvoie si nombre_a_tester est de la forme 2k ; 4k + 1 ; 16k + 3"""
    return (nombre_a_tester % 2 == 0) or (nombre_a_tester % 4 == 1) or (nombre_a_tester % 16 == 3)


def verifier_tous_les_nombres():
    """Test tous les nombres entiers naturels et vérifie s'ils vérifient la conjecture de Syracuse"""
    nombre_courant = 1
    while True:
        if not est_exclu(nombre_courant):
            sequence_a_tester = suite_de_syracuse(nombre_courant)
            if sequence_a_tester[-1] == 1:
                print(f"{nombre_courant} vérifie la conjecture de Syracuse !")
            else:
                print(f"{nombre_courant} ne vérifie pas la conjecture de Syracuse !")
        nombre_courant += 1
```

Par des calculs analogues, nous prouvons prouver que plus de 90% des nombres entiers naturels sont inutiles à tester car nous prouvons prouver que leur vol en altitude est fini. Ainsi, seuls les nombres impairs de la forme $16k + 7, 16k + 11$ et $16k + 5$ peuvent être des prétendants à un contre exemple. En effet, nous ne pouvons pas prouver avec exactitude que ces vols atterissent car leur comportement varie en fonction des nombres.

II. Page d'accueil index.html

1) Informations

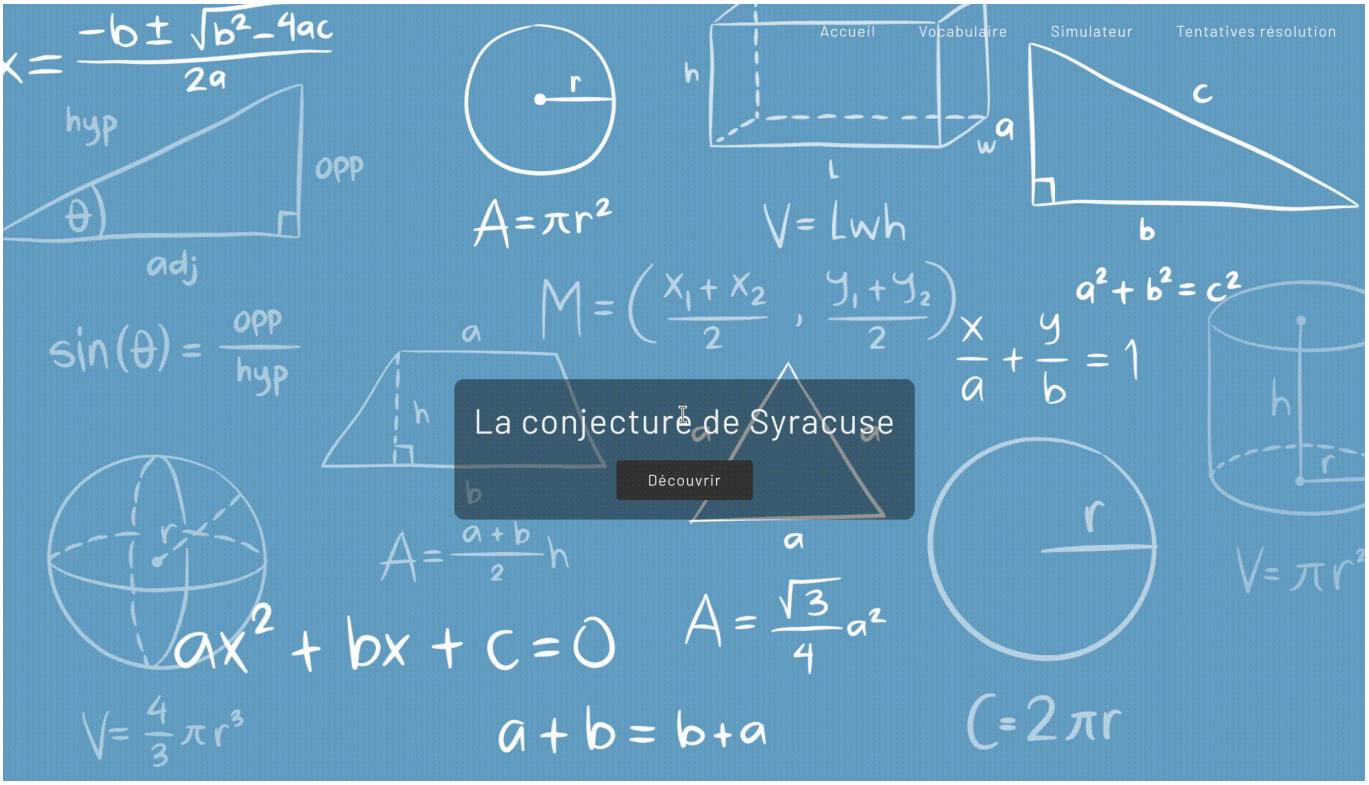
Cette page sert de page d'accueil et de page principale du site.
Dans celle-ci sont expliquées ou introduites les notions les plus fondamentales de la conjecture

- Les origines et les divers noms
- L'application de Syracuse
- La définition rigoureuse de Syracuse par récurrence
- La définition de la suite de Syracuse compressée, très utilisée notamment dans la page tentatives-resolutions.html

Remarque : Tous les points sont illustrés par des graphiques que j'ai créé à l'aide du simulateur de la page simulateur.html

2) Style

Cette page utilise des sections de classe presentation. De plus, au scroll de la page ou au clique sur le bouton découvrir, une légère animation d'apparition progressive des sections de présentation se lance :



Pour ce faire, le scroll est détectée grâce à un écouteur d'événement appliqué à la fenêtre tout entière. La fonction apparitionSection() est alors appelée. Celle-ci parcourt l'ensemble des sections grâce à une boucle forEach. Pour chaque section de la page, sa position sur celle-ci est calculée par rapport au haut de la fenêtre et si cette position est située à moins de 100px de la portion visible de l'écran, alors la classe visible est appliquée à la section. Cette classe est définie en css pour faire apparaître progressivement l'élément grâce à TranslateY et opacity.

III. Page d'explication du vocabulaire voca.html

1) Informations

Cette page sert à définir les notions plus complexes relatives à la conjecture de Syracuse :

- Le vol d'une suite de Syracuse
- Le temps de vol d'une suite de Syracuse
- L'altitude maximale d'une suite de Syracuse
- Le temps de vol en altitude d'une suite de Syracuse

Cette page utilise des sections de classe presentation et présente un style plutôt similaire à index.html

IV. Page de simulateur simulateur.html

1) Description

Cette page est certainement la page la plus importante du site. En effet, celle-ci permet de :

- Taper n'importe quel nombre entier strictement positif et de pouvoir observer un graphique représentant sa suite de Syracuse sous différents repères : linéaire ou semi-logarithmique. Le premier est le repère par défaut lors de la génération du graphique et permet de respecter la physionomie de la suite, passant de très haut pic à des pics très bas. Cependant, pour des valeurs élevées, celui-ci peut devenir rapidement illisible, c'est pourquoi il est possible de changer pour un repère semi-logarithmique couvrant des ordres de grandeur variés en utilisant une échelle logarithmique sur un axe et une échelle linéaire sur l'autre
- Obtenir des informations plus précises sur la suite que l'on a générée, comme le nombre d'itération nécessaire pour atteindre 0, la valeur maximale atteinte ou encore le facteur d'expansion.
- De télécharger la suite de Syracuse du nombre généré sous forme d'un fichier .csv facilement importable dans Excel par exemple pour pouvoir exploiter les données Il est également possible de faire un clic droit sur le graphique généré et de l'enregistrer sous format .png

2) Style

Cette page reprend les éléments graphiques basiques du site comme le fond bleu, la barre de navigation... Cependant, hormis les éléments cités plus haut, la page est complètement indépendante d'un point de vue de mise en page et possède donc un fichier css dédié : `simulateur.css` rangé dans le dossier `css`. Au chargement, la page se présente très sobrement, n'étant composée que d'un titre, d'une barre d'entrée et d'un bouton. Or, la page est composée de trois autres sections : `ensemble-btn`, `zone-graphique` et `informations` qui sont par défaut en `display: none;`. Au clic sur le bouton de validation, si l'entrée est validée, ces trois sections passent en `display: block` ou `display: flex` dans le cas de la div `ensemble-btn`. Par cette action, effectuée en javascript, une zone blanche comportant le graphique voulu apparaît au centre de l'écran avec au dessus deux boutons permettant de changer le repère de ce dernier et en bas une zone permettant de découvrir quelques informations sur la suite du nombre tapé, comme son nombre d'itération, sa valeur maximale d'expansion et son facteur d'expansion. Si l'entrée n'est pas validée, donc si l'entrée n'est pas un entier strictement positif, une `alert` apparaît afin d'indiquer à l'utilisateur son erreur.

3) Fonctionnement

Le fonctionnement du générateur de graphique est assuré par deux fichiers javascripts : `app.js` et `fonction.js`. Le fichier `fonction.js` sert à définir les fonctions utilisées pour faire fonctionner l'application. On y trouve : - La fonction `syracuseSuite` qui, pour tout nombre passé en argument, nous renvoie un objet contenant la suite de Syracuse, le nombre d'itération et le maximum atteint - La fonction `supprimerGraphiqueExistant` qui sert à supprimer le graphique déjà généré sur la page. Cette fonction est très utile dans le cas où l'utilisateur veut changer de repère ou souhaite simplement générer le graphique d'un autre nombre sans avoir à recharger la page - La fonction `creerGraphique` qui sert de pièce centrale dans le bon fonctionnement de l'application. Celle-ci prend trois arguments: 'ctx`, `infos` et `repere`. Le but de cette fonction est de pouvoir créer un moule le plus générique possible qui pourra s'adapter au nombre entré ou aux actions de l'utilisateur grâce aux arguments que l'on spécifie à l'appel de la fonction. En effet, plutôt que de copier coller le code nécessaire pour générer un graphique (qui est assez massif) pour au final ne modifier que la ligne du repère par exemple, il est plus judicieux de simplement supprimer le graphique préexistant et d'appeler la fonction de nouveau avec `"logarithmic"` comme valeur de l'argument `repere`. Nous échangeons donc 52 lignes pour une seule. - La fonction `verifierEntree` qui sert à vérifier si l'entrée saisie par l'utilisateur est bien un nombre et que celui-ci est strictement positif. Sans cette vérification, l'utilisateur pourrait entrer n'importe quoi comme des chaînes de caractère ou des nombres décimaux ou négatifs, ce qui poserait problème lors de la mise en graphique de la suite. La page se figerait alors et l'utilisateur n'aurait d'autres choix que de l'actualiser. - La fonction `exporterCSV` qui permet d'exporter les données générées sur la suite de Syracuse du nombre tapé (grâce à la fonction `syracuseSuite`) sous un format `suite_de_syracuse_de_.csv`. Elle prend ces données en argument sous forme d'un objet. Dans un premier temps, la fonction modifie la séquence pour y ajouter des `\n` permettant un saut de ligne entre chaque valeur et rendant la lecture plus agréable lorsque le fichier csv est importé. Puis, un fichier est créé avec la méthode `encodeURI` avec un encodage `utf-8`.

Le second fichier, appelé app.js, sert à relier les actions de l'utilisateur et les fonctions définies au préalable dans le fichier fonction.js. Y sont effectuées les tâches suivantes :

- La déclaration de constantes reliée au document html grâce à la méthode getElementById
- La gestion du changement de couleur des boutons changeant le repère. En effet, le repère sélectionné devient bleu pendant que le second vire au gris. Pour ce faire, on parcourt chacun des boutons de la div et on y ajoute un écouteur d'événement sur le clic grâce à addEventListener. Au clic sur le bouton (1), le bouton (2) est d'abord désactivé puis le bouton (1) prend sa couleur bleu signifiant qu'il est celui qui a été cliqué
- L'écouteur d'événement sur le clic principal, celui gérant les actions lors du clic de l'utilisateur sur le bouton de validation de l'entrée. Voici les étapes réalisées dans l'ordre pour générer le graphique :

- On récupère l'entrée tapée avec la méthode .value appliquée sur la barre d'entrée.
- On vérifie que l'entrée satisfaie bien les conditions établies dans le fonction verifierEntree. Si ce n'est pas le cas, on affiche un message avec alert indiquant à l'utulisateur que son entrée est invalide
- Si l'entrée est valide, on récupère le contexte du canvas défini en html, on génère l'objet contenant les informations sur la suite de Syracuse du nombre entré, on vérifie qu'aucun graphique ne se trouve sur la zone voulue avec la méthode creerGraphique, puis l'on crée un nouveau graphique creerGraphique(ctx, infos, "linear");. Par défaut, le répère est linéaire, ce qui explique que le bouton correspondant soit directement coloré lors de la première génération.
- On vérifie à l'aide d'une variable drapeau que le clic de l'utilisateur sur le bouton de validation est bien le premier. Si tel est le cas, on définit les trois sections de telle sorte qu'elles deviennent visibles à l'écran comme dit précédemment. Cette variable drapeau nous sert à ne pas appliquer un style à un élément l'ayant déjà.
- On crée deux écouteurs d'événement sur les boutons servant à changer le repère et on détruit puis régénère un nouveau graphique correspondant
- On remplit les informations comme le maximum, le nbr d'itération ou le facteur d'expansion grâce aux données stockées dans l'objet
- On crée un écouteur d'événement sur le bouton permettant de télécharger les informations en csv Diviser ces tâches en deux fichiers distincts permet une meilleure lisibilité et maintenabilité du code car si nous voulons modifier une fonction ou une action lors du clic, nous savons directement quel fichier gère quoi.

V. Page tentatives-resolutions.html

Cette page sert à regrouper et à expliquer différentes tentatives pour vaincre la conjecture de Syracuse. Ainsi, les approches étudiées sont :

- L'approche inverse via la conception étape par étape de l'arbre de Syracuse
- La recherche de cycles différents du cycle (4, 2, 1) via des raisonnements par l'absurde permettant de montrer l'unicité du cycle trivial parmi ceux de longueurs trois, la non existence de cycles de longueurs 5 et l'énonciation du théorème de Eliahou
- L'approche algorithmique visant à évincer des cas qui ne sont pas prétendants à un contre exemple et à présenter un programme python permettant de tester tous les nombres entiers tout en excluant les cas évincés
- La mise en place d'un indice probabiliste visant à expliquer le caractère plutôt décroissant de la suite au moyen de calculs probabilistes
- Une extension sur Z de la suite de Syracuse qui nous permet de constater que la conjecture de Syracuse ne tient pas à l'extension aux nombres relatifs
- La présentation d'une variante de la suite de Syracuse

Cette page utilise des sections de classe carte permettant de laisser plus de place au texte explicatif tout en permettant d'intégrer une image illustrative si besoin

VI. Remarque finale

Le site n'est pas exhaustif, et il n'a par ailleurs pas vocation à l'être. Le but était avant tout, par le biais de ce projet, de m'instruire sur un sujet que je trouve très intéressant. Ainsi, j'ai tout du long privilégié les informations que je comprenais, les informations sur ce sujet faisant paprfois appel à des notions avancées de mathématique. Je voulais par exemple intégrer une explication sur l'extension aux nombres réels de la suite de Syracuse. Cependant, au fur et à mesure de l'écriture, je me rendais compte que je recopiais plus que je ne comprenais. En effet, je ne pense pas, par exemple, totalement maîtriser les notions de point fixe attractif et répulsif malgré des lectures sur le sujet. Même constat pour l'extension aux nombres complexes, entremêlant des notions que je connaissais (toutes les bases du plan complexe, l'ensemble de Julia, l'ensemble de Mandelbrot et les figures fractales associées) avec des notions plus obscures (notion de topologie comme ensemble borné, fermé ou connexe ; la notion de voisinage...). En conclusion, choisir ce sujet m'a permis de découvrir de nouveaux horizons mathématiques et de mettre en oeuvre quelques notions que j'ai appris du site Mathtraining