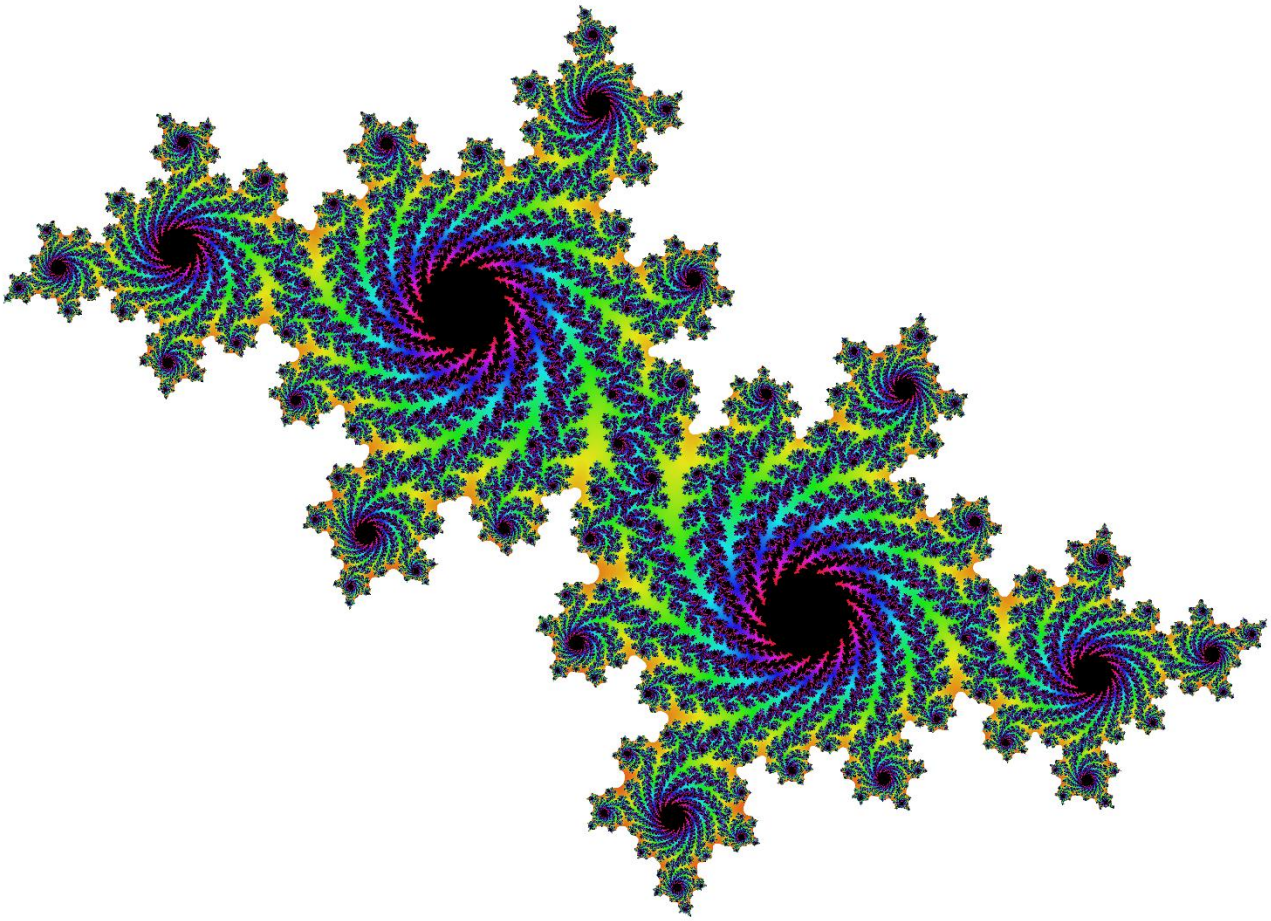
LES TROPHÉES NSI

Édition 2023

DOSSIER DE CANDIDATURE
PRÉSENTATION DU PROJET



Ce document est l'un des livrables à fournir lors du dépôt de votre projet : 4 pages maximum (hors documentation).

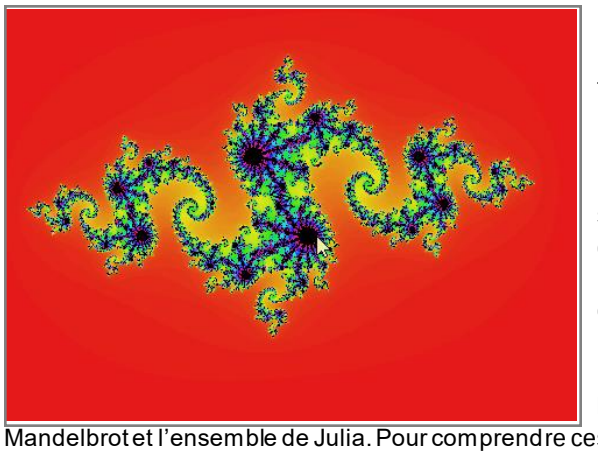
Pour accéder à la liste complète des éléments à fournir, consultez la page **Préparer votre participation**.

Vous avez des questions sur le concours ? Vous souhaitez des informations complémentaires pour déposer un projet ? Contactez-nous à **info@trophees-nsi.fr**.

NOM DU PROJET : EzFractal

> PRÉSENTATION GÉNÉRALE :

Les fractales sont des objets mathématiques qui ont la propriété d'être autosimilaires, c'est-à-dire que leur structure est répétée à différentes échelles. En d'autres termes, si l'on zoome sur une partie d'une fractale, on peut voir des motifs similaires à ceux observés à une plus grande échelle.



Dans le projet, nous aimons tous les deux les sciences et plus précisément les maths et l'informatique et nous trouvons les fractales très fascinantes visuellement et par leurs fonctionnements.
Dans le cadre de l'enseignement de la spécialité NSI nous devions créer un projet en groupe. Nous nous sommes donc dit que de proposer un petit logiciel simple qui permettra d'explorer les fractales serait un bon moyen de faire découvrir ce que sont les fractales et que cela serait un projet intéressant pour progresser dans le monde du développement.
Nous nous sommes intéressés aux fractales générées à l'aide de nombres complexes qui sont l'ensemble de Mandelbrot et l'ensemble de Julia. Pour comprendre ces fractales, il faut d'abord savoir ce que sont les nombres complexes, ce sont des nombres composés d'une partie réelle et d'une partie imaginaire ce qui fait que l'on peut les placer dans un repère à 2 axes. Tout comme avec les réels, nous pouvons faire des opérations avec les nombres complexes. Ensuite, pour notre fractale, nous utilisons une suite sous la forme $Z^2 + c$ ou Z et c sont tous deux des nombres complexes. Avec n'importe quelle valeur de Z ou de c si nous itérons la suite un grand nombre de fois il y a 2 possibilités, soit la suite diverge vers l'infini soit elle boucle entre plusieurs valeurs (Ex : 0,1,0,1,0,1).

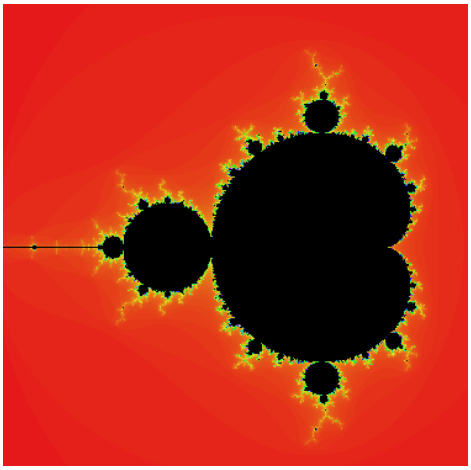
Une fois que nous savons cela, nous allons nous intéresser comment est généré l'ensemble de Julia :

- On définit la constante de Julia qui correspond à c (Ex : c = -1 + 0i)

Pour chaque pixel :

- Nous déterminons la valeur de Z en fonction des coordonnées x et y du pixel.
- Ensuite, nous itérons la suite $Z^2 + c$
- Si la suite n'a pas divergé avant un nombre maximum d'itération, on définit le pixel en noir.
- Sinon en fonction du nombre d'itérations que prend la suite à diverger, on définit une couleur pour le pixel en question.

Voilà comment fonctionne l'ensemble de Julia, il y a un nombre énorme d'alternatives car il y a une infinité de possibilités pour la valeur que l'on donne a c.



Ensuite, pour générer la fractale de Mandelbrot, on reprend le même fonctionnement seulement Z est forcément défini à 0 et c est défini en fonction des coordonnées de chaque pixel.

Il y a un fort lien entre les deux ensembles, on retrouve notamment des ensembles de Julia dans l'ensemble de Mandelbrot car celui-ci est une sorte de carte des ensembles de Julia.

L'ensemble de Mandelbrot et plus généralement toutes les fractales sont fascinantes. Il y a encore plein de choses à découvrir et à apprendre ainsi que certaines choses qui n'ont pas encore été découvertes.

Dans les choses intéressantes, on peut noter la relation entre Mandelbrot et le nombre π, la suite de Fibonacci ou encore les fractales de Newton.

On retrouve aussi le fait que ces 2 ensembles ont une dimension bien spécifique. On peut aussi modéliser l'ensemble de Julia dans de plus grandes dimensions comme en 4D à l'aide de quaternion.

> ORGANISATION DU TRAVAIL :

- SMOLENSKI Robin : Je me suis occupé de comprendre le fonctionnement des fractales et comment nous pourrions les générer à l'aide de Python. Ensuite, j'ai cherché des idées de fonctionnalités qui pourraient être intéressantes. Nous avons mis nos idées en commun puis j'ai designé les interfaces graphiques à l'aide de logiciels tels que Figma. Tout au long du projet, j'ai donné des idées d'amélioration à la fois dans le code et dans l'expérience utilisateur.

- RUHF Quentin : Je me suis occupé de la partie programmation et de bien implémenter les fonctionnalités. J'ai commencé avec l'aide de mon camarade à faire une première version d'implémentation des fonctions de génération des fractales. Ensuite, je me suis occupé de réaliser les interfaces utilisateur qui sont faites avec Ez (wrapper qui se base sur Pygame) et JSON pour les styles. Pour les interfaces utilisateur, j'ai voulu créer un système inspiré du fonctionnement C#/XAML en séparant la partie logique et la partie du style. Pour cela, j'ai utilisé la POO (Programmation Orientée Objet). J'ai créé des composants pour chaque type comme une case à cocher, une vue défilante, un bouton, etc. Je me suis inspiré des composants que l'on retrouve en XAML avec .NET. Une fois tous les composants créés, il m'a suffi de remplir un fichier JSON avec tous les composants et les styles de chaque composant que j'ai récupéré directement sur Figma. Grâce à la POO et quelques scripts, les composants se chargent automatiquement quand on modifie le JSON.
Enfin, j'ai implémenté toutes les fonctionnalités et je les ai optimisés.

Nous avons utilisé un dépôt GitHub où le projet a été mis à jour à chaque fois. Pour la communication, nous avons notamment utilisé Discord.
Nous avons travaillé durant quelques heures de cours au lycée et le reste a été fait à la maison. La partie du code a demandé plus de temps, proche de 100 heures voir plus, mais cela nous passionne.

LES ÉTAPES DU PROJET :

Etapes	Description	Difficultés
Idées	Nous avons commencé par chercher les idées et les différentes fonctionnalités. On note dans un fichier Markdown les différentes interfaces, les touches à utiliser et les fonctionnalités. Nous avons aussi défini les caractéristiques voulues comme les langages et les plateformes visées (Windows et Linux).	Pas de difficultés particulaires mise à part à définir le maximum d'idées au début pour ne pas se compliquer la tâche à implémenter des fonctionnalités par la suite.
Labo test	Ensuite, nous avons créé un projet de test, nous avons fait les premières fonctions qui permettent de générer les fractales, ces premières fonctions étaient très peu performantes mais fonctionnelles. Nous avons essayé plein de choses pour voir ce qui était faisable et comment nous pourrions implémenter les différentes choses pour que cela fonctionne en symbiose (UI qui communique avec la logique, etc.). Le but était de se donner un premier aperçu de comment développer le logiciel et de ne pas se lancer sans rien. Cela nous a aussi permis de mieux comprendre le fonctionnement des fractales.	Une des premières difficultés après avoir réussi à générer notre première fractale a été les performances. La fractale prenait plusieurs secondes à se générer ce qui rendait l'exploration impossible. Pour cela, nous avons dû trouver de nombreuses stratégies comme l'utilisation d'un tableau de pixels pour tracer à l'écran à la place de dessiner pixel par pixel. Ensuite, nous avons utilisé Numpy pour améliorer les performances et enfin, la plus grande des optimisations a été l'utilisation de Numba pour compiler les fonctions qui génèrent les fractales. Ce qui a rendu l'exploration des fractales possible passant de 0 image par seconde a environ 25 images par seconde.

Modification de la lib EZ	Dans le cadre de notre spécialité NSI, nous avons été contraints d'utiliser la bibliothèque EZ créée par notre professeur qui a pour but de simplifier Pygame. Cela nous a posé quelques problèmes. La bibliothèque étant écrite intégralement en français, nous avions pour but de créer notre projet entièrement en anglais pour l'accessibilité et se rapprocher du monde du développement professionnel et de l'open source. Ensuite, il manquait quelques fonctionnalités qui n'étaient pas présente dans la bibliothèque. Nous avons donc réécrit toute la bibliothèque en anglais ainsi que rajouter et modifier des fonctionnalités. Nous avons gardé un maximum la structure de base de cette bibliothèque.	Pas de grande difficulté, il a juste fallu comprendre la façon dont la bibliothèque a été créer.
Créations des Composants	Ensuite, nous avons cherché une manière de créer les UI, nous nous sommes inspirés du fonctionnement de .NET avec XAML et C#. Nous avons donc créé à l'aide de la POO des composants comme les boutons, les vues défilantes, les champs de caractères, les boutons a bascule etc. Chaque composant a des propriétés telles que la position x, y, taille etc. Toutes ces propriétés sont stockées dans des fichiers JSON et sont chargées dynamiquement.	Le composant de la vue défilante a été un assez gros problème, cela fonctionne avec un tableau de pixel où on affiche seulement une partie du tableau. L'affichage et le fonctionnement de la barre défilante a été difficile.
Créations des UI	Une fois les composants créés, la création des UI a été très facile, il a suffi de créer un fichier JSON avec toutes les instances des composants ainsi que leurs propriétés. La plupart d'entre eux tels que la position et la taille sont donnés par Figma donc notre maquette nous a été très utile. Il a ensuite suffi de rajouter quelques morceaux de code pour la logique des composants.	Pas de difficultés particulaires.
Optimisation	Nous avons ensuite fait quelque optimisation en spécifiant le type de quelques variables ainsi que l'améliorer de la structure et l'organisation du projet. Nous avons utilisé les Github actions avec CodeQL ainsi que DeepSoruce pour faire de l'analyse de projet afin de trouver d'éventuelles failles et d'optimiser le code. De plus, nous avons utilisé les outils de « code profiler » et de « code coverage » intégrer à notre IDE qui est PyCharm Professional.	La bibliothèque EZ n'étant pas directement typée cela nous a poser des problèmes. Il a fallu examiner chaque fonction de EZ puis regarder sur la documentation Pygame.
Correction de Bug	Nous avons corrigé quelques bugs tels qu'un problème avec le clavier azerty, qwerty ou encore des bugs d'affichage.	Pas de difficultés particulaires.
Formatage du code	Nous avons effectué un formatage du code avec black un formateur python pour respecter une certaine rigueur et propreté du code. Nous avons aussi optimisé le code en simplifiant des expressions et les imports.	Pas de difficultés particulaires.
Finalisation	Nous avons créé une documentation complète sur GitHub et mis le projet sous licence GPL v3.	Pas de difficultés particulaires.

> FONCTIONNEMENT ET OPÉRATIONNALITÉ :

• Le logiciel est divisé en plusieurs « applications » qui sont les différent menu ou nous pouvons naviguer. Chaque « application » est un objet python il a donc des propriétés, les plus importante sont le screen_array qui est un tableau numpy qui contient chaque pixel de l'écran, c'est-à-dire les valeurs RGB de chaque pixel. Par exemple screen_array[10][0] correspond au dixième pixel sur la première ligne et vaudra [255, 255, 255] si le pixel est blanc. Ensuite, nous avons un objet UI attaché à chaque « application » cet objet a le rôle de modifier le tableau de pixel en fonction des composants a afficher et des événements utilisateur. L'objet application a une fonction nommée run qui va créer une fenêtre pygame et ensuite dans une boucle infinie, elle va appeler la fonction de l'objet UI pour mettre à jour le tableau de pixel en fonction des événements puis, elle va afficher ce tableau dans la fenêtre. Voici un schéma simplifier du fonctionnement :

```python
class Export_App:
    Wokia-Dev
    def __init__(self, launcher, app):
        self.resolution = width, height
        self.launcher = launcher
        self.application = app
        self.screen_array = np.full((width, height, 3), [255, 255, 255], dtype=np.uint8)
        self.export_app_ui = ExportUI(self.application)

    Wokia-Dev
    def run(self, from_return: bool = False):
        EZ.create_window(self.resolution[0], self.resolution[1], caption,
                         self.application.working_directory + "/Resources/Images/icon.png")
        EZ.change_cursor(pygame.SYSTEM_CURSOR_ARROW)
        self.export_app_ui.run()
        while True:
            EZ.draw_array(self.screen_array)
            self.export_app_ui.run()
            self.export_app_ui.check_events()
            EZ.update()
```

Le tableau de pixel.

Le fichier UI attaché à l'application.

On affiche le tableau de pixel dans la fenêtre.

On appelle une fonction de l'objet UI pour mettre à jour le tableau de pixel.

• Les fonctions permettant de générer les fractals opèrent directement sur le tableau de pixel afin d'optimiser les performances. De plus, elles sont compilées à l'aide de numba. Voilà par exemple notre fonction pour les fractals de Julia (la fonction pour Mandelbrot est sensiblement la même) :

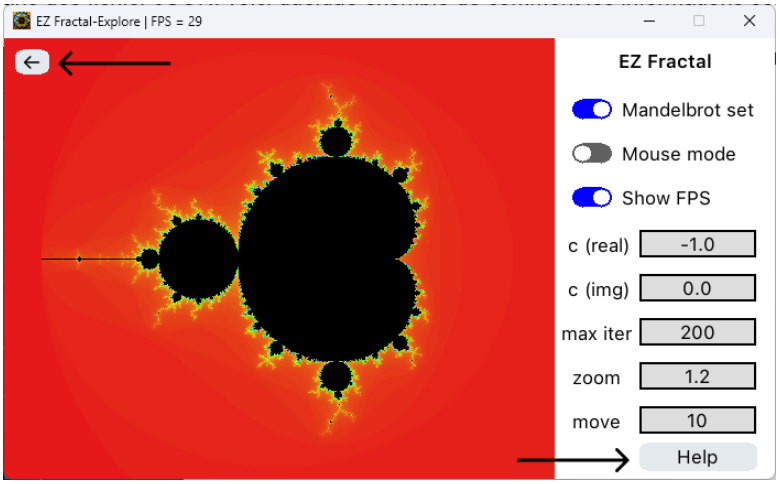
```python
@numba.njit(fastmath=True, parallel=True)
def render_julia(
        screen_array: np.array,
        c: complex,
        max_iter: int,
        zoom: float,
        offset: np.array,
        width: int,
        height: int,
        menu_width: int = 0,
        saturation: float = 0.8,
        lightness: float = 0.5,
):
    # foreach pixel in the screen array using numba parallel
    for x in numba.prange(width - menu_width):
        for y in numba.prange(height):
            # define the complex number based on the pixel coordinates, zoom and offset
            z = (x - offset[0]) * zoom + 1j * (y - offset[1]) * zoom
            # number of iterations
            num_iter = 0

            # iterate the function until the number is diverging or the max iterations is reached
            for i in range(max_iter):
                # julia set formula
                z = z ** 2 + c
                if z.real ** 2 + z.imag ** 2 > 4:
                    # exit the loop if the number is diverging
                    break
                num_iter += 1

            # define the color based on the number of iterations and set the pixel color in the screen array
            screen_array[x, y] = iter_gradient_generator(
                num_iter, max_iter, saturation, lightness
            )
    # return the screen array
    return screen_array
```

Les fonctions de générations sont détaillées davantage dans la documentation GitHub ou les algorithmes sont expliqués.

• Pour les interfaces utilisateur, les propriétés de chaque composant sont stockées en liste d'objets dans des fichiers JSON, voici quelques exemples de comment les informations sont stockées :


```json
"EzButtons": [
  {
    "name": "btnHelp",
    "x": 577,
    "y": 367,
    "width": 106,
    "height": 25,
    "text": "Help",
    "background_color": "E3E9EC",
    "background_opacity": 255,
    "font_size": 17,
    "font_color": "000000",
    "font_family": "SF-Pro-Text-Regular",
    "border_radius": 7,
    "text_margin": [
      -18,
      -11
    ],
    "click_timer": 300
  },
  {
    "name": "btnReturn",
    "x": 10,
    "y": 10,
    "width": 30,
    "height": 22,
    "text": "\u2190",
    "background_color": "E3E9EC",
    "background_opacity": 255,
    "font_size": 20,
    "font_color": "000000",
    "font_family": "SF-Pro-Text-Regular",
    "border_radius": 7,
    "text_margin": [
      0,
      0
    ],
    "click_timer": 300
  }
],
```


Voici l'exemple des 2 boutons du menu d'exploration, on n'y retrouve un identifiant, les positions, la taille, etc.



Ensuite chaque composant correspond à un objet python où l'on retrouve toute ces propriétés ainsi qu'une fonction qui afficher le composant en fonction de ces paramètres :

```python
class EzButton(EzComponent):
    """EzButton class for creating buttons"""

    deepsource-autofix[bot] +1
    def __init__(
        self,
        name: str,
        x: int,
        y: int,
        width: int,
        height: int,
        text: str,
        background_color: str,
        background_opacity: int,
        font_size: int,
        font_color: str,
        font_family: str,
        border_radius: int,
        text_margin: list[int],
        click_timer: int,
    ):
        super().__init__(name, x, y, width, height)
        self.text = text
        self.background_color = background_color
        self.background_opacity = background_opacity
        self.font_size = font_size
        self.font_color = font_color
        self.font_family = font_family
        self.border_radius = border_radius
        self.text_margin = text_margin
        self.click_timer = click_timer
```

> OUVERTURE :

Nous sommes assez contentes, car notre logiciel fonctionne, mais le logiciel est peu performant et aurait pu être plus joli. Pour améliorer cela si nous avions eu le choix nous aurions utilisé un langage et des bibliothèques plus adapter, nous aurions sûrement fait cela avec .NET et WPF avec le langage C# et XAML ou avec QT et le langage C++ ou encore avec la SDL. Le projet nous a quand même permis de se familiariser avec python et surtout de découvrir les fractals.
Pour ce qui est de la diffusion du projet mis à part le dépôt open source sur GitHub, nous ne tenons pas forcément à plus le diffuser. Mais il me semble quand même qu'une version améliorée de ce projet serait intéressante à diffuser sur YouTube par exemple pour bien expliquer le fonctionnement et faire découvrir les fractals et la création d'un tel projet du point de vue technique.
Si nous devions refaire ce projet, je pense que nous passerions plus de temps sur la structure du projet pour permettre de rendre plus performant et surtout sur la partie des composants je pense que nous essayerons de développer cela d'avantage pour pouvoir créer une sorte de bibliothèque graphique a l'image de QT par exemple pour le rendre accessible dans d'autres projets bien que cela sera sûrement beaucoup moins puissant que des bibliothèques telles que QT.
Une amélioration qui pourrait être assez simple et de rajouter d'autre fractals comme les fractales de Newton, le Burning Ship ou encore essayer de modéliser des fractals en 4D à l'aide des quaternions bien que cette dernière amélioration demande beaucoup de connaissance mathématique et une grande puissance de calcul.

DOCUMENTATION

Spécifications Techniques

- Le logiciel a été pensé, développer et tester pour les plateformes Windows et Linux, il n'a pas été tester sur MacOs, mais cela devrait fonctionner avec quelques modifications mineures.
- Le logiciel nécessite une version de python supérieur à 3.0, nous conseillons la version 3.10. Vous pouvez installer python via ce lien.
- Les bibliothèque python utilisé sont Pygame, Numpy et Numba. Pygame est utiliser pour la partie graphique, Numpy et Numba sont utiliser pour l'optimisation des performances, Numba nous permet des compiler certaines fonctions python ce qui améliore grandement la performance.

Installation

Pour commencer, il vous faudra télécharger le logiciel pour cela, vous pouvez utiliser git en ligne de commande (CLI). Ensuite, il vous faudra entrer dans le dossier :

```
1 git clone https://github.com/Wokia-Dev/EzFractal.git
2 cd EzFractal
```

Ensuite, il vous faudra installer les dépendances python pour cela, il vous suffira d'utiliser la commande pip :

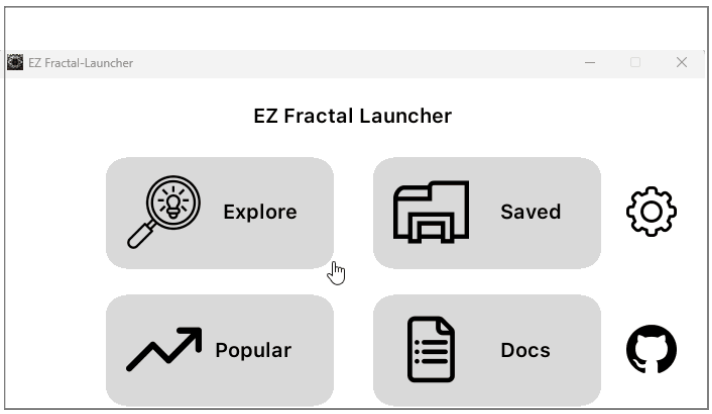
```
1 pip install -r requirements.txt
```

Pour finir, il vous suffira d'exécuter le fichier principal à l'aide de python :

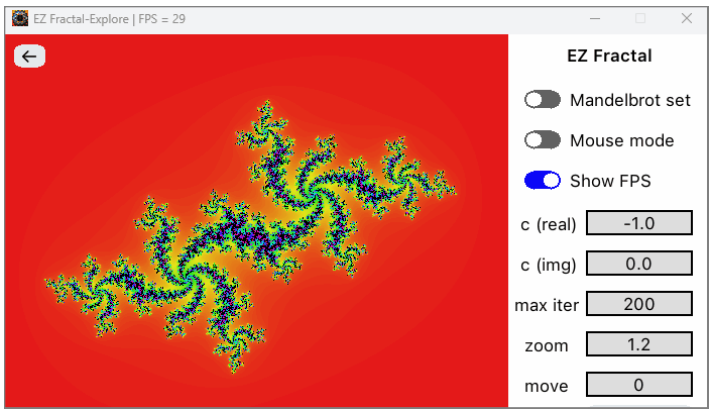
```
1 python main.py
```

Utilisation (les images qui suive sont des gifs vous pouvez cliquer dessus pour les lires)

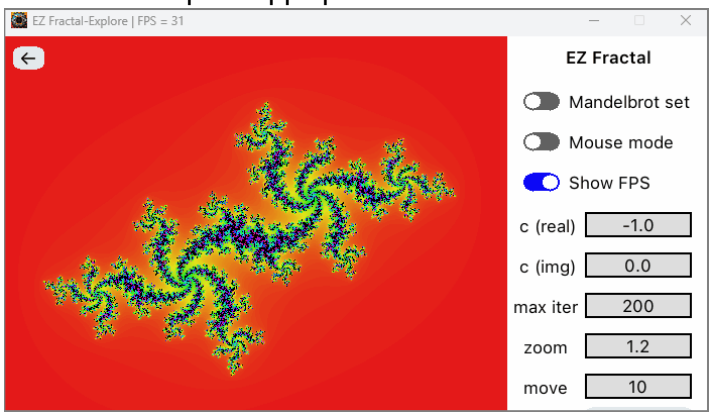
• Vous pouvez naviguer entre les différents menus simplement en cliquant sur les boutons ou le bouton de retour en arrière :



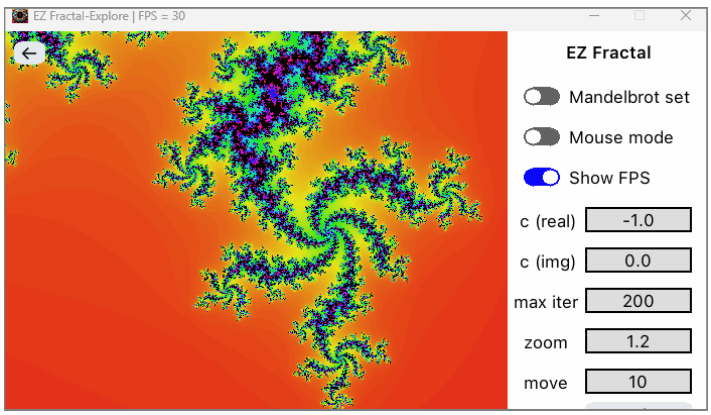
• Pour naviguer dans la fractals vous pouvez utiliser les flèches directionnelles pour se déplacer (ou les touches ZQSD si les paramètres ont été modifier), la molette de la souris pour zoomer et dézoomer, enfin pour réinitialiser à la position de départ, vous pouvez utiliser la touche R :



• Pour modifier les paramètres de la fractale à droite, il vous suffit simplement de cliquer pour les boutons et pour les champs de caractères veillez à laisser la souris par-dessus le champ de texte ensuite, il vous suffira de taper au clavier les valeurs souhaitées puis la touche entrer pour appliquer la valeur :



• Quand nous le souhaitons, nous pouvons exporter une image, il vous suffit d'utiliser le raccourci ctrl + s ensuite il vous suffira de choisir le facteur de la taille de l'image et de l'exporter. Plus le facteur de taille sera élevé, plus l'exportation sera longue car le logiciel fait les calculs sur une plus grande surface :



Une autre documentation est aussi disponible sur notre [dépôt GitHub], cette documentation a l'avantage d'être disponible à la fois en français et en anglais. De plus, elle dispose d'une meilleure mise en forme ce qui peut la rendre plus agréable à lire ainsi qu'une petite vidéo de démonstration.

Images du logiciel