README.MD :

- conseils d'installation
- informations de compilations et lancement du projet

Projet.pdf :

Rédaction intégrale du projet :

- Explications des L-systèmes,
- Interprétation graphique des L-Systèmes
  - La tortue
  - Interprétation des chaînes
  - Evolution de l'interprétation des chaînes des L-Systèmes
  - Remarques sur les parcours des itérations

- Implémentation des L-Systèmes en Ocaml
  - Le sujet minimal
  - Choix des types et code fourni
    - Tortue
    - Chaînes parenthésées
    - Substitutions
    - Interprétations
    - L-Systèmes
    - Exemples des L-Systèmes
  - Extensions
    - Sauvegarde de L-Systèmes ou d'images obtenues
    - Couleurs
    - Variations de paramètres
    - Graphique en trois dimensions
- Modalités
  - Dates limites de soumission et dates de soutenances début Janvier.
  - Projet sur le serveur GitLab de l'UFR d'Informatiques
- Usage de GitLab
  - Accès au serveur et configuration personnelle
  - Création du dépôt
  - Création et synchronisation des copies locales de travail
  - Modification du dépôt : les commits
  - Les fusions(merge) et les conflits
  - Intégrer les modifications venant du dépôt du cours
  - Les branches

Références au livre : *The algorithmic beauty of plants*, de **Prusinkiewicz et Lindenmayer.**

Makefile :

Un **makefile** est déjà présent, il permet de compiler en binaire comme en byte, en passant par « **dune** » avec « **build** ». '**make**' compile le projet en code natif. '**make byte**' compile en bytecode, notamment pour faire tourner le code en toplevel Ocaml. '**make clean**' permet de supprimer les fichiers intermédiaires du projet, notamment ceux construits par dune lors de la compilation.
'**./run ...**' est la commande pour exécuter le projet.

Dune et dune-project
la configuration de dune pour le projet. Notons que la version 1.6 dans dune-project a été modifié par la **version 1.2** par nos soins.

Run
fichier exécutable. Exécution avec dune.

Lsystems.top
Lance un script qui lance un **toplevel OCaml** avec tous les modules du projet excepté **main.ml**. Pour ce script une compilation en byte est nécessaire. Permet de tester le code sur un toplevel.

Les fichiers de code :
Il y en a 7, 4 modules ocaml, et 3 interfaces de module :
Les interfaces :

- **examples.mli** : initialise un type symbol avec trois symboles, pour l'exemple d'un snow flake. Un exemple est consiste en un axiome, un système de réécritures et une interprétation.

- **systems.mli** : initialisation de trois type :
    - **'s word** composé d'un **'s** appelé '**Symb**', d'une liste de **'s word** appelé '**Seq**', et d'un **'s word** appelé '**Branch**'. C'est la représentation de l'arbre syntaxique d'une chaîne parenthésée. Symb est le type choisi pour les symboles. **Seq** représente une suite ordinaire de sous-chaînes. **Branch** représente une chaîne entre crochets.
    - **'s rewrite_rules** qui à un **'s** renvoie un **'s word** : représente la substitution, associe des symboles à des chaînes parenthésées.
    - **'s system** composé d'un '**axiom**' de type **'s word**, de '**rules**' de type **'s rewrite_rules**, de '**interp**' de type **'s** renvoyant **Turtle.command list** . C'est la représentation d'un L-Système. **axiom** est le dictionnaire de mots décrivant le système. **Rules** est la règle de substitution. **Interp** est l'interprétation « graphique » du système.
- **turtle.mli** : possède la commande graphique de la tortue ainsi qu'un type position donnant la position exacte de la tortue :
    - **command** est un type qui peut être un entier(**Line** qui trace un trait, **Move** qui bouge sans tracer de traits, ou **Turn** qui change l'angle de direction de la tortue), **Store** ou **Restore**(respectivement pour enregistrer la position courante de la tortue, et pour restaurer la dernière position sauvegardé pas encore restauré).
    - Il y a le type position également avec des nombres flottants **x** et **y**(coordonnées abscisses et ordonnées de la tortue) et un entier **a** pour son angle d'orientation.

Les modules :

- **examples.ml** : fonction construisant un **symbol snow** en **system** pour le fameux « snow flake »
- **systems.ml :** identique à **systems.mli** il y aura dans ce fichier toutes les fonctions implémentés concernant les systèmes.
- **turtle.ml** : identique à **turtle.mli** il y aura dans ce fichier toutes les fonctions implémentés concernant les tortues.
- **main.ml** : s'occupe de la gestion des arguments de la ligne de commande interprétés par le module **Arg** de OCaml. C'est dans ce fichier qu'est indiqué le programme en lui-même. Par

ce fichier toutes les fonctions nécessaires au fonctionnement du L-Système sont appelés dans un ordre précis au bon fonctionnement du programme pour que celui ci agisse comme nous l'attendons.

Examples :

Ce répertoire contient 9 L-systèmes que nous pouvons retrouver dans le livre mis en référence de **projet.pdf** ou sur **Wikipedia.** Ces exemples sont similaire à **example.mli**/**example.ml** sont écrits dans des fichiers **.sys** telles qu'ils soient faciles à analyser. L'explication est claire dans la **dernière sous-partie du 3.2** de **projet.pdf.**