RAPPORT DU PROJET DE FIN DE SEMESTRE

Spécialité : **2éme année de Licence Informatique**

Module : **Base de la Programmation Fonctionnelle**

Préparé au sein de : UFR des Sciences & Techniques

CRÉATION D'UN MODULE DE TRAÇAGE DES COURBES DE JULIA

Présenté et soutenu par :
Ismail TAHLI

Les Outils Utilisés

- Éditeur de Code : Visual Studio Code
- Version d'Ocaml : Ocaml version 4.08.1

Les Sources d'Aides

- StackOverFlow : En cherchant des solutions à des problèmes similaires aux problèmes rencontrés.
- Github : En s'inspirant des codes de Julia_Set rédigés en C et en JavaScript.
- Comprendre le calcul des ensembles de Julia : https://mathcurve.com/fractals/julia/
- Module Complex d'Ocaml : https://caml.inria.fr/pub/docs/manual-ocaml/libref/Complex.html
- Module Scanf d'Ocaml : https://caml.inria.fr/pub/old_caml_site/ocaml/htmlman/libref/Scanf.html
- http://www-sop.inria.fr/lemme/Olivier.Pons/htdocs/ENSEIGNEMENT/OBJET/OCAML/cours1.html

Durée du projet

Le Projet a été fait INDIVIDUELLEMENT dans 3 Jours seulement sans demander de l'aide à personne sachant qu'avant les vacances je ne disposais pas de matériels informatiques (Ordinateur) et que j'avais beaucoup de retard à récupérer surtout en TP, mais une fois que j'ai acheté un PC Portable,(la 2éme semaine des vacances),j'ai commencé le projet.

Les Problèmes Rencontrés

- Le premier problème rencontré est la compréhension de l'énoncé.
- Je ne savais presque rien sur le Module Graphique d'Ocaml.
- Après m'être renseigné sur les boucles while/for d'Ocaml, j'ai essayé d'écrire les fonctions lvl_of et julia en utilisant des instructions itératifs mais j'ai rencontré un problème car la valeur rendu par ces boucles doit être modifiée à chaque itération et en Ocaml nous avons toujours travaillé que avec des variables constantes et le mot clef « let » ne fait pas l'affection. Alors en cherchant une solution je suis tombé sur une notion de référence « ref » de la bibliothèque standard avec des opérateurs (!) qui permet d'accéder au contenu de la référence et (:=)qui permet de donner une nouvelle valeur a son contenu (Voir L'image N° 1).
- Transformer toutes les boucles itératives à des fonctions récursives terminales.
- J'ai trouvé des difficultés à comprendre l'utilisation de la fonction **sscanf** du module **Scanf**, surtout qu'il n'y pas trop d'explication sur le fonctionnement de cette dernière, et des difficultés aussi avec les types de formatages de cette fonction.
- La fonction read_line() dans notre programme prend seulement deux entrées, la chaîne de caractère qui désigne la commande et un réel qui désigne la nouvelle valeur à mettre dans la configuration, mais le problème c'est que le changement du nombre complexe « c » doit avoir deux arguments (la partie réelle et la partie imaginaire). Ce problème a été résolu en appelant la fonction read_float() pour entrer la partie imaginaire du nombre complexe si la chaîne de caractère désigne bien un changement de « c » .
- Le changement de la dimension de la fenêtre ne s'effectue pas parce qu'il effectue le changement sur une fenêtre déjà ouverte.

  La solution: Fermer cette dernière après chaque modification de la configuration et la rouvrir avec la nouvelle dimension.

Le rôle de chaque partie du programme

✓ Type config : c'est un type enregistrement dont les champs définissent les éléments configurables du programme et ils sont mutables pour qu'on puisse les modifiés en cas de besoin.

✓ Type command : c'est un type somme énumérant les différentes commandes de changement de la configuration et contenant la nouvelle valeur à modifier dans une configuration.

✓ La fonction check de type (string → float→ command): prend en paramètre deux arguments (s : une chaîne de caractère et n :un réel), et qui effectue un filtrage de « s » et associe cette dernière à la commande correspondante avec la nouvelle valeur. Pour le changement du nombre complexe nous faisons un autre appelle à la fonction read_float() pour saisir sa partie imaginaire.

✓ La fonction read_command de type (unit → command) : permet de lire une commande depuis l'entrée standard, en faisant appel à deux fonction : read_line() qui lit sur l'entrée standard et la fonction sscanf du module Scanf qui effectue un formatage de ce qui a été lu et elle lui associe une fonction en l'occurrence dans notre cas la fonction check.

✓ La fonction execute de type (config → command → config) : modifie une configuration en fonction d'une commande et retourne la nouvelle configuration avec une fermeture et une réouverture de la fenêtre pour que le changement soit bien lisible.

✓ La fonction lvl_of de type (conf → Complex.t → int):prend deux paramètres (une configuration et un complexe) et renvoie un entier, plus exactement (calcule p(n) avec n le plus petit n ≤ borne tel que sa norme soit supérieur à 2 et elle le renvoie s'il existe, la valeur borne sinon).

✓ La fonction julia de type (conf → unit) : prend un seul paramètre (une configuration) et renvoie un élément de type unit, son rôle est de colorer tous les pixels de la fenêtre graphique, plus exactement (pour chaque point p de R, elle réalise le calcul de son niveau de gris en utilisant fonction « lvl_of » et affiche le point p avec le niveau de gris correspondant).

✓ La fonction récursives terminale read_command_and_draw de type (conf → unit) : son rôle est de lire une commande sur l'entrée standard, d'appliquer la modification à la configuration puis de dessiner la courbe de Julia correspondante.

✓ La fonction principale main de type (config → unit) : prend une configuration en paramètre, puis ouvre une fenêtre graphique, trace la courbe de Julia associée à la configuration et lance la boucle d'interaction « read_command_and_draw » qui est récursive.

✓ La fonction openf de type (config → unit) : prend une configuration en paramètre et ouvre une fenêtre avec les dimensions de cette dernière pour pour éviter la duplication de code.

La notice d'utilisation

L'utilisation du programme est facile.

- D'abord nous compilons le {projet}.ml si ce n'est pas déjà fait, et pour ça nous dirigeons vers le dossier contenant le code et nous ouvrons le terminal dans ce dernier et nous exécutons la fonction suivante : (Voir l'image 2)
  ocamlc -o {bytefile} graphics.cma {projet}.ml
- Nous exécutons le programme dans le même dossier avec la commande : (Voir l'image 3)
  ./{bytefile} ou ocamlrun bytefile
- Le programme commence par dessiner une courbe de Julia avec une configuration déjà fournie dans le code source. (Voir l'image 3)
- Pour changer la configuration (*Dessiner une nouvelle courbe, changer la dimension, changer le repère…*) nous faisons appel à la commande souhaitée sur l'entrée standard.
- Pour changer la dimension de la fenêtre : nous faisons appel à la commande suivante sur l'entrée standard et nous cliquant sur entrée: (Voir l'image 4)
  ResizeWindow NouvelleValeur
- Pour changer le repère ou la zone de dessin : nous faisons appel à la commande suivante sur l'entrée standard et nous cliquant sur entrée: (Voir l'image 5)
  ChangeRepere NouvelleValeur
- Pour changer le zoom : nous faisons appel à la commande suivante sur l'entrée standard et nous cliquant sur entrée: (Voir l'image 6)
  ChangeZoom NouvelleValeur
- Pour changer la borne de n: nous faisons appel à la commande suivante sur l'entrée standard et nous cliquant sur entrée: (Voir l'image 7)
  ChangeBorneNouvelleValeur
- Et finalement pour changer nombre complexe « c »: nous faisons appel à la commande suivante avec la partie réelle de ce dernier sur l'entrée standard puis nous cliquant sur entrée ↵ et finalement nous saisissons la partie imaginaire du même nombre complexe et nous cliquant à nouveau sur entrée ↵ pour afficher la nouvelle courbe : (Voir l'image 8)
  ChangeComplex PatieReel ↵
  PartieImaginaire