

# Rapport du projet de programmation Impérative: Stackchess

Loïc DUBARD

30 avril 2019

## Table des matières

<b>1</b>	<b>Phase 1</b>	<b>2</b>
1.1	Démarche générale . . . . .	2
<b>2</b>	<b>Fonctionnement général du script</b>	<b>2</b>
<b>3</b>	<b>Tests</b>	<b>2</b>
<b>4</b>	<b>Idées d'optimisation du code</b>	<b>2</b>

# Introduction

## DERNIÈRES NOUVELLES : LES EXTRA-TERRESTRES EXISTENT

Le but de ce projet est d'implanter un protocole de communication avec les habitants des mondes lointains.

Plus précisément, il s'agira de prendre un message entrée et de donner la suite d'instructions permettant de piloter une (ou plusieurs) antenne(s) afin d'envoyer ledit message.

Dans l'urgence de cette nouvelle, nous avons développé un système d'antennes rudimentaire composées chacune d'une roue comportant les 26 lettres de l'alphabet et du caractère espace. Nous pouvons donc voir l'ensemble des symboles utilisables comme un cycle. Le caractère suivant A est B, ..., le caractère suivant Y est Z, le caractère suivant Z est espace suivi de A.

Toutes les antennes sont initialement positionnées sur le caractère espace.

Les commandes disponibles pour manipuler une antenne sont au nombre de 3 :

- "N" : passe au caractère suivant.;
- "P" : revient au caractère précédent;
- "E" : envoie la commande d'émission du caractère courant.

Nous devons commencer rapidement à envoyer des messages afin d'ouvrir au plus vite la voie à nos ingénieurs financiers et autres analystes de données. Le projet sera donc découper en trois phases (une antenne, plusieurs antennes et compression du signal).

Durant toutes ces phases vos programme devront lire deux lignes sur leur entrée standard.

la première ligne comportera un entier  $n$  donnant le nombre d'antennes disponibles. Pour la première phase, cet entier sera toujours égal à 1. la deuxième ligne comportera le message lui-même.

Le programme affichera donc une suite de commandes permettant d'émettre le message.

## 1 Phase 1

### 1.1 Démarche générale

Le principe est le suivant : je défini la roue comme une liste des caractères (en majuscule) :

```
let roue = [' ','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R';  
           'S','T','U','V','W','X','Y','Z']
```

Puis, je demande à l'utilisateur le message à traduire grâce aux fonctions [parse\_input ()] et [split\_string s] fournies par le professeur.

Le couple ( $n$ :int, $l$ :char list) ainsi obtenu est passé dans la fonction [commande ( $n,l$ ) roue].

On introduit alors la fonction [translate c roue] qui se charge de renvoyer la commande correspondant à l'envoi de la lettre  $c$ :char, sous forme de string. (Il suffit pour cela de récupérer l'indice (noté  $i$ ) du caractère  $c$  dans roue et de renvoyer  $i$  fois 'N' si  $i < 14$  sinon  $27-i$  fois 'P').

On introduit aussi la fonction [tourner c n roue]

Cette nouvelle fonction agit par récurrence sur la liste de caractères  $l$  de cette manière :

- à la première itération, on traduit le premier caractère de la liste en sa commande (une string) grâce à la fonction translate qui renvoie un couple ()
- à la  $n^{i\text{eme}}$  iteration, le  $n^{i\text{eme}}$  caractère est traduit en sa commande (en une string) en utilisant la roue

## **2 Fonctionnement général du script**

## **3 Tests**

## **4 Idées d'optimisation du code**

Au lieu de choisir la roue à utiliser lettre après lettre, on peut tracer un arbre  $n$ -aire (où  $n$  est le nombre de roues) et trouver le chemin le plus court pour la phrase à traduire. On optimiserai le temps d'exécution du programme non pas lettre par lettre mais pour toute la phrase.

Cependant, cette manière de calculer le programme à envoyer aux antennes me semble très couteuse en temps et en mémoire. C'est pour cela que j'ai choisi de ne pas l'implémenter.