

# Participation à l'ICFP 2015

## 1 Introduction

Marc de Falco a proposé durant l'été aux membres de l'UPS de participer au tournoi de programmation de l'ICFP<sup>1</sup> (voir [icfpcontest.org](http://icfpcontest.org)) qui s'est tenu sur trois jours du vendredi 7 août 2015 (14h) au lundi 10 août 2015 (14h). À son appel, une équipe (TaupeGoons) s'est formée pour relever le défi. Elle était constituée de quatre membres :

- Marc de Falco, le GO du groupe qui a tout orchestré et a abattu la majorité du travail ;
- Jean-Julien Fleck, qui s'est occupé de l'optimisation des paramètres par algorithme génétique ;
- Laurent Jospin, qui malheureusement a dû déclarer forfait suite à des soucis informatiques ;
- Loïc Pottier, qui a développé son propre moteur de résolution et a fini par former sa propre équipe (GaupeToons), d'abord pour le tester, puis pour essayer de l'améliorer puisque les résultats étaient plutôt bons.

Le présent article vise à décrire le problème posé pour le tournoi ainsi que la manière dont l'équipe TaupeGoons s'est ingénié à le résoudre. Tout le code informatique utilisé peut être retrouvé à l'adresse

<https://github.com/MarcdeFalco/icfp15>

## 2 Problème posé

### 2.a Description rapide

Le problème à résoudre consiste à écrire un programme jouant tout seul à TETRIS® sur une grille hexagonale.

L'ensemble des pièces à utiliser est propre à chaque plateau et la pièce à poser est déterminée par un générateur pseudo-aléatoire fixé.

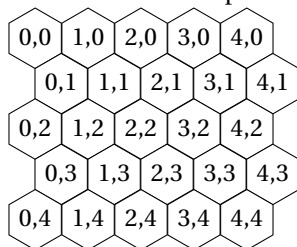
Chaque pièce posée de manière définitive (dans la suite on dira qu'elle a été *verouillée*) permet de marquer des points.

Chaque mouvement possible d'une pièce peut être représenté par plusieurs caractères et la solution doit être présentée sous la forme d'une chaîne de caractères.

---

1. International Conference on Functional Programming

FIGURE 1.1: Système de coordonnées pour la grille hexagonale



L'ambiguïté sur la représentation d'un mouvement permet l'inclusion de mots intelligibles dans une solution tout en préservant son sens et son impact sur le plateau.

Dix-huit phrases spéciales peuvent être incluses dans les solutions pour gagner des points. Il s'agit donc de jouer au mieux tout en faisant apparaître le plus de phrases spéciales dans la solution.

## 2.b Plateau et système de coordonnée

Le plateau est une grille hexagonale bidimensionnel de cellules.

Une cellule a deux états : *libre* ou *occupée*.

Les rangées de la grille sont disposées en quinconce et on en déduit le système de coordonnées suivant présenté dans la figure 1.1

Notons tout de suite que ce système de coordonnées a le problème suivant : les déplacements dans la grille ne sont pas proportionnel à des déplacements géométrique. En effet, si on souhaite passer effectuer une translation d'un cran en bas à droite (ce qui sera défini comme le mouvement  $\searrow$  au paragraphe 2.f) on ne peut pas se contenter d'ajouter (1, 1) aux coordonnées car si le point (4, 7) devient effectivement (5, 8) après cette translation, le point (4, 6) devient (4, 7). Tout dépend de la parité de la rangée sur laquelle un point se situe.

## 2.c Pièces

Une pièce est déterminée par un ensemble de cases qu'elle occupe en position neutre et la donnée d'un point sur la grille appelé son pivot, qui n'est pas nécessairement une des cases occupées et dont les coordonnées peuvent sortir du plateau.

Voici des exemples de pièces où les cases vides sont colorées et le pivot est indiqué par un cercle au centre de la cellule sur laquelle il se situe :



## 2.d Données initiales

Chaque instance du problème à résoudre est décrit par un sextuplet  $(w, h, \mathcal{O}, N, \overline{g}, \overline{p})$  où  $w \times h$  les dimensions du plateau

$\mathcal{O}$  l'ensemble des cases déjà occupées

$N$  le nombre de pièces à placer lors de chaque partie

$\overline{g} = (g_1, \dots, g_m)$  les germes des parties à jouer

$\overline{p} = (p_1, \dots, p_n)$  les différents types de pièces disponibles.

## 2.e Générateur pseudo-aléatoire

Afin de déterminer la pièce courante un générateur pseudo-aléatoire est utilisé avec un germe donné dans la description de l'instance.

On commence par définir la suite congruente linéaire  $(u_n)_{n \in \mathbb{N}}$  où  $u_0$  est le germe et

$$\forall n \in \mathbb{N}, u_{n+1} = (1103515245u_n + 12345) \mod 2^{32}$$

Le générateur pseudo-aléatoire  $v_n$  correspond aux bits 16 à 30, en commençant à numéroter 0 le bit de poids faible, de la suite  $u_n$ . Soit avec une formule mathématique :

$$\forall n \in \mathbb{N}, v_n = \left\lfloor \frac{u_n}{2^{16}} \right\rfloor \mod 2^{15}$$

## 2.f Mouvements et verrouillage

Pour déplacer une pièce on dispose de six mouvements :

- ← mouvement d'un cran à gauche (Ouest)
- mouvement d'un cran à droite (Est)
- ↙ mouvement d'un cran à gauche et d'un cran en bas (Sud-Ouest)
- ↘ mouvement d'un cran à droite et d'un cran en bas (Sud-Est)
- ⌚ rotation d'un angle  $\frac{\pi}{3}$  autour du pivot (Rotation anti-horaire)
- ⌚ rotation d'un angle  $-\frac{\pi}{3}$  autour du pivot (Rotation horaire)

Un mouvement est valide si toutes les cellules occupées après celui-ci sont libres.

Lorsqu'on effectue un mouvement invalide la pièce est verrouillée et les cellules qu'elle occupe sont marquées comme étant occupées.

La figure 1.2 présente une succession de mouvements aboutissant à un verrouillage.

## 2.g Suppression de ligne

Lorsque après avoir verrouillé une pièce des rangées du plateau sont entièrement occupées elles sont remplacées par des cases libres et les rangées situées au dessus descendent d'un cran (les cases des rangées paires effectue donc un mouvement ↙ et les autres un mouvement ↘).

Un exemple de ce mécanisme est présenté à la figure 1.3.

## 2.h Déroulement d'une partie

Partant d'une instance  $(w, h, \mathcal{O}, N, \overline{g}, \overline{p})$  du problème et d'un germe  $g_i$ , une partie se déroule ainsi :

1. On initialise le générateur pseudo-aléatoire avec le germe  $g_i$ .
2. On initialise le plateau avec une grille de dimension  $w \times h$  et on marque comme étant occupées les cases données par l'ensemble  $\mathcal{O}$ .

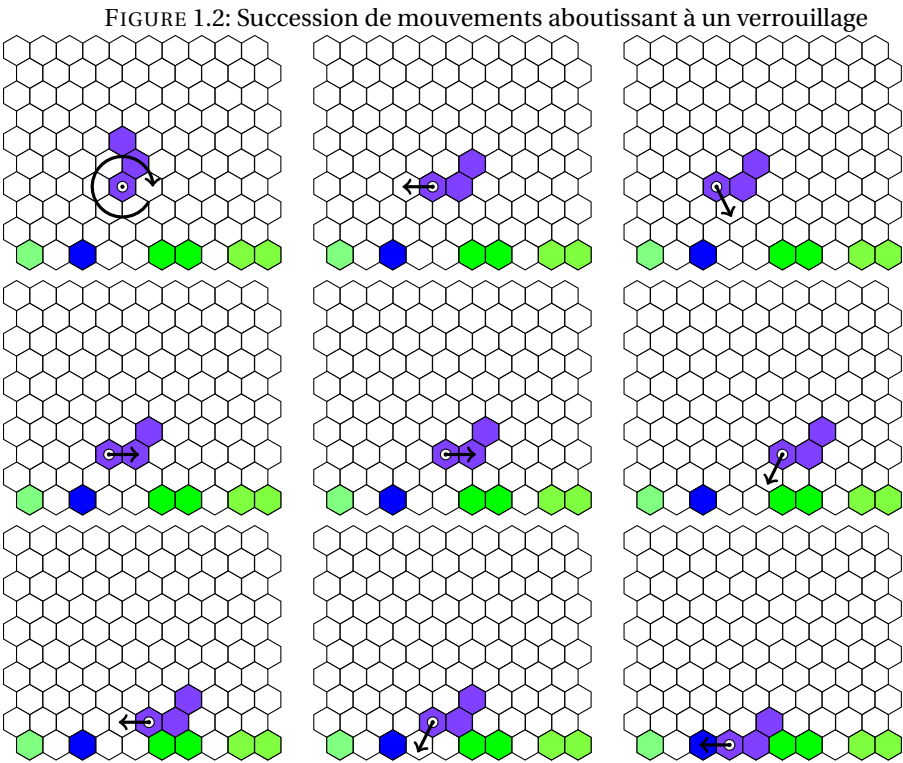
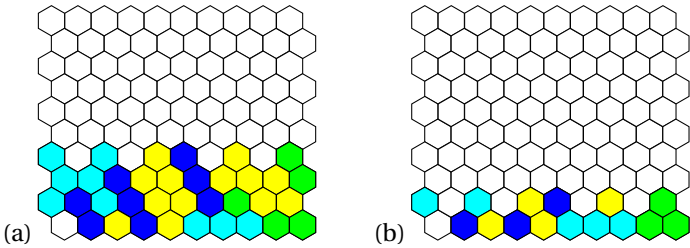


FIGURE 1.3: Disposition d'un plateau avant (a) et après (b) l'étape de suppression de lignes



### 3. Tant qu'on n'a pas placé $N$ pièces

- i) Soit  $k$  généré par le générateur pseudo-aléatoire et  $k' = k \bmod n$  où  $n$  est le nombre de pièces dans  $\overline{p}$ .
- ii) On génère une nouvelle pièce  $p_{k'}$  en la plaçant au sommet du plateau et en la centrant horizontalement.
- iii) Si cette position n'est pas valide, la partie s'arrête.
- iv) On effectue alors une suite de mouvements aboutissant à un verrouillage de la pièce.
- v) Si lors d'un mouvement une pièce a son pivot et les cases qu'elle occupe dans une position déjà prise dans les mouvements précédents, la partie s'arrête.
- vi) Si nécessaire on supprime les lignes pleines.

Il y a donc trois conditions d'arrêt :

- soit les  $N$  pièces ont été placées ;
- soit une pièce n'a pas pu être générée au sommet du plateau ;
- soit une pièce a été placée deux fois dans la même position.

La figure 1.4 présente un exemple de partie.

## 2.i Encodage d'une solution

Une solution à une partie est une chaîne de caractères encodant la suite de mouvements à effectuer où chaque mouvement peut être encodé par plusieurs caractères selon la correspondance présentée dans la figure 1.5.

## 2.j Phrases spéciales

Dans la mesure où il est possible d'utiliser plusieurs caractères pour encoder un mouvement, une même solution peut s'exprimer d'un grand nombre de manières, certaines d'entre elles faisant apparaître des mots compréhensibles.

Une part importante du score final pour une partie est déterminé par la fréquence d'apparition de phrases spéciales décrites à la figure 1.6.

On peut remarquer que certaines de ces phrases sont longues et nécessitent beaucoup de cases libres pour être jouées sans coups invalides. Lorsqu'une pièce a certaines symétries de rotation vis-à-vis de son pivot, elles peuvent être toujours invalides. Par exemple pour une pièce triviale réduite à une seule case contenant son pivot aucune des phrases contenant des rotations n'est valide.

Lors des trois jours du concours les phrases spéciales étaient cachées, pour les obtenir il fallait résoudre des devinettes apparaissant au fil du déroulement. Notre équipe a découvert 15 des 18 phrases spéciales<sup>2</sup>.

---

2. Dont 5 dans la dernière heure grâce à des échanges avec d'autres équipes...

FIGURE 1.4: Un exemple de partie avec (a) la disposition du plateau en début de partie, (b) les différents types de pièces et (c) une disposition en fin de partie



FIGURE 1.5: Correspondance entre les mouvements et les caractères

mouvement	caractères
←	p'!.03
→	bcefy2
↙	aghi j4
↘	lmno 5
↻	dqrvz1
↺	kstuw x



### 3 Solution choisie

#### 3.a Système de coordonnées obliques et rotations

Il existe un système de coordonnées géométriques simple pour une grille hexagonale : le *système de coordonnées obliques* qui est présenté à la figure 1.7.

Ce système correspond aux coordonnées dans la base  $(\vec{u}, \vec{v}) = \left( (1, 0), \left(\frac{1}{2}, -\frac{\sqrt{3}}{2}\right) \right)$  de  $\mathbb{R}^2$  pour des hexagones de diamètre 1. Pour passer du système originel vers celui-ci il suffit d'appliquer la transformation

$$(x, y) \xrightarrow{\text{vers oblique}} \left( x + \left\lfloor \frac{y}{2} \right\rfloor, y \right)$$

et pour revenir dans le système originel il suffit de faire la réciproque dans la mesure où la seconde coordonnée est inchangée

$$(x, y) \xrightarrow{\text{vers originel}} \left( x - \left\lfloor \frac{y}{2} \right\rfloor, y \right)$$

L'avantage de ce système est immédiat quand on considère l'impact des mouvements de descente sur les coordonnées comme présenté à la figure 1.8.

Mais là où ce système est particulièrement intéressant c'est qu'il permet d'effectuer des rotations d'angle  $\frac{k\pi}{3}$ ,  $k \in \mathbb{Z}$ , en utilisant uniquement des opérations arithmétiques entières<sup>3</sup>. Partant d'un point de coordonnées  $(x, y)$  dans la base canonique de  $\mathbb{R}^2$ , on sait qu'en effectuant une rotation de centre l'origine et d'angle  $\frac{-\pi}{3}$  on aboutit au point de coordonnées  $(x', y')$  où

$$x' = \frac{x + \sqrt{3}y}{2} \quad y' = \frac{-\sqrt{3}x + y}{2}$$

On considère maintenant le point de coordonnées obliques  $(x_o, y_o)$  qui a donc pour coordonnées dans  $\mathbb{R}^2$  le couple  $(x_o + \frac{y_o}{2}, -\frac{\sqrt{3}}{2}y_o)$ . Après rotation, on obtient le couple

$$\left( \frac{x_o + \frac{y_o}{2} - \sqrt{3}\frac{\sqrt{3}}{2}y_o}{2}, \frac{-\sqrt{3}x_o - \frac{\sqrt{3}y_o}{2} - \frac{\sqrt{3}}{2}y_o}{2} \right) = \left( \frac{x_o}{2} - \frac{y_o}{2}, \frac{-\sqrt{3}}{2}(x_o + y_o) \right) = -y_o\vec{u} + (x_o + y_o)\vec{v}$$

Pour effectuer un mouvement de rotation horaire dans le système oblique on obtient ainsi la transformation élémentaire :

$$(x, y) \xrightarrow{\circlearrowright} (-y, x + y)$$

et en procédant de même on obtient

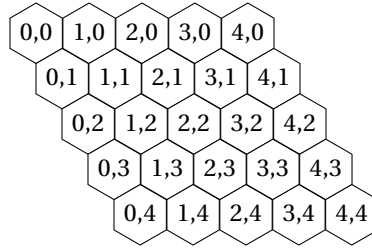
$$(x, y) \xrightarrow{\circlearrowleft} (x + y, -x)$$

Notons qu'il est possible d'en déduire une formule de rotation dans le système originel, ce qui donne pour la rotation horaire :

$$(x, y) \xrightarrow{\text{vers oblique}} \left( x + \left\lfloor \frac{y}{2} \right\rfloor, y \right) \xrightarrow{\circlearrowright} \left( -y, x + \left\lfloor \frac{y}{2} \right\rfloor + y \right) \xrightarrow{\text{vers originel}} \left( -y + \left\lfloor \frac{x + \left\lfloor \frac{y}{2} \right\rfloor + y}{2} \right\rfloor, x + \left\lfloor \frac{y}{2} \right\rfloor + y \right)$$



FIGURE 1.7: Système de coordonnées obliques

FIGURE 1.8: Effet d'un mouvement sur les coordonnées  $(x, y)$  selon le système de coordonnées

Mouvement	Coordonnées problème	Coordonnées obliques
←	$(x - 1, y)$	$(x - 1, y)$
→	$(x + 1, y)$	$(x + 1, y)$
↙	$\begin{cases} (x - 1, y + 1) & \text{si } y \text{ pair} \\ (x, y + 1) & \text{sinon} \end{cases}$	$(x - 1, y + 1)$
↘	$\begin{cases} (x, y + 1) & \text{si } y \text{ pair} \\ (x + 1, y + 1) & \text{sinon} \end{cases}$	$(x, y + 1)$

On peut alors choisir d'utiliser ces coordonnées uniquement pour calculer les positions après transformation ou les utiliser partout en ne revenant aux coordonnées initiales que pour accéder au plateau. Lors du concours nous avons fait le premier choix car le code avec déjà été écrit sans les rotations qui sont arrivées trop tard pour partir sur le bon système de coordonnées.

Notons un autre avantage du système de coordonnées obliques : la translation est géométrique, donc un simple ajout aux coordonnées. Cela donne la possibilité d'identifier une pièce à la position de son pivot et de sa rotation tout en pouvant calculer très simplement la position de ses cellules. En effet, on peut calculer toutes les rotations d'une pièce au départ puis se contenter de translater les cellules.

### 3.b Identification unique des positions

Afin de pouvoir assurer qu'aucune répétition de position a lieu il ne suffit pas de prendre en compte des couples  $(M, r)$  où  $M$  est la position du pivot et  $r$  la rotation effectués sur la pièce.

En effet, certaines pièces sont invariantes par certaines rotations et la seule donnée de  $r$  ne permet pas de le prendre en compte. On considère pour cela son groupe d'invariants.

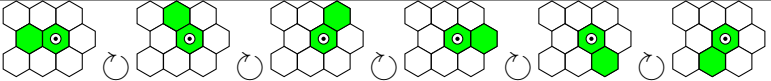
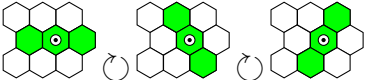
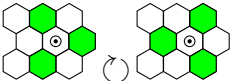
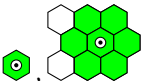
Soit  $p$  une pièce on note  $r_p$  la rotation d'un angle  $\frac{\pi}{6}$  autour de son pivot.

On note  $R(p)$  le sous-groupe des isométries du plan engendrées par  $r_p$  et  $I(p)$  le sous-groupe de  $R(p)$  constitué des éléments laissant  $p$  invariante.

Pour une position donnée du pivot, on a donc autant de possibilités pour la pièce que d'éléments dans  $G_r(p) = R(p)/I(p)$  (tous ces groupes sont abéliens).

3. On pourrait objecter que le passage d'un système de coordonnées à l'autre nécessite une division, mais comme il s'agit d'une division par deux, un simple décalage à droite l'effectue, cette dernière opération étant encore plus élémentaire qu'une addition.

FIGURE 1.9: Les quatre cas pour le groupe  $G_r(p)$ 

$G_r(p) \sim \mathbb{Z}/k\mathbb{Z}$	Exemple
$k = 6$	
$k = 3$	
$k = 2$	
$k = 1$	

Comme  $R(p) \sim \mathbb{Z}/6\mathbb{Z}$  on a quatre cas pour  $G_r(p)$  qui sont illustrés dans la figure 1.9.

On peut alors représenter uniquement la position d'une pièce à symétrie près par un couple  $(M, r)$  où  $M$  est la position du pivot et  $r \in G_r(p)$ .

En pratique, on identifie  $r$  à l'élément  $k \in [0; 5]$  tel que  $r$  soit associé à  $\bar{k}$  par les isomorphismes ci-dessus.

On est donc ramené à un triplet d'entier par position. Comme le pivot peut sortir de la zone de jeu, il faut border le tableau pour pouvoir tenir compte de ces positions du pivot.

Un calcul rapide sur les pièces disponibles permet de déterminer une valeur  $b$  telle que le pivot dans une zone de jeu de dimension  $w \times h$  ait des coordonnées dans  $[[-b; w+b-1]] \times [[-b; h+b-1]]$ .

Grâce à cet encodage on peut créer un tableau de booléens à trois dimensions  $V$  tel que  $V_{x,y,k}$  indique si la position où le pivot est en  $(x, y)$  et la rotation est celle associée à  $k$  est visitée.<sup>4</sup>

Pour maintenir la valeur de  $k$  à jour lors des mouvements, il suffit de remarquer qu'une rotation revient à ajouter  $\pm 1$  dans  $G_r(p)$ .

### 3.c Parcours en largeur pour découvrir les positions verrouillables

Pour placer une pièce, notre solution commence par énumérer les positions verrouillables. Pour cela, on part de la pièce en position initiale puis on effectue tous les mouvements possibles pour découvrir de nouvelles positions qu'on maintient dans une file. Ensuite on procède de même en enlevant un élément dans la file tant qu'elle est non vide.

Pour ne pas visiter plusieurs fois une même position on maintient un tableau de booléen indexé par l'identifiant unique vu en 3.b.

Quand à partir d'une position il n'est pas possible d'effectuer tous les mouvements, alors il existe un mouvement permettant de verrouiller la pièce. On peut alors ajouter cette position aux positions verrouillables.

Cela nous donne le pseudo-code suivant :

4. Durant le concours, en raison de la présence du bord  $b$  à calculer et des problèmes potentiels, nous avons uniquement utilisé une liste de positions visitées ce qui est bien entendu plus couteux mais plus sûr lorsque l'on dispose de peu de temps.

```

verouillables = []
visitées = crée tableau de taille W x H x 6 initialisé à faux
à_visiter = file vide

ajoute position_initiale à à_visiter

Tant que à_visiter est non vide:
    position = enleve à à_visiter

    Si visitées[position]:
        passer à la suite

    visitées[position] = vrai

    mouvements_qui_verouillent = []

    Pour chaque mouvement:
        nouvelle_position = effectue mouvement depuis position
        Si nouvelle_position est valide:
            ajoute nouvelle_position à à_visiter
        Sinon:
            ajoute mouvement à mouvements_qui_verouillent

    Si mouvements_qui_verouillent est non vide:
        mouvement = mouvements_qui_verouillent[0]
        ajoute (position, mouvement) à verouillables

```

En faisant ainsi on obtient en fin d'algorithme la liste des positions verrouillables et il n'est pas compliqué de maintenir la suite de mouvements qui nous a permis d'atteindre cette position depuis la position initiale.

Pour ne pas garder des positions verrouillables inintéressantes notre algorithme ne gardait pas les positions au dessus de la dernière cellule occupée sur le plateau.

On obtient alors le nouveau pseudo-code :

```

verouillables = []
visitées = crée tableau de taille W x H x 6 initialisé à faux
à_visiter = file vide

ajoute (position_initiale, chemin_vide) à à_visiter

Tant que à_visiter est non vide:
    position, chemin = enleve à à_visiter

    Si visitées[position]:
        passer à la suite

```

```
visitées[position] = vrai

mouvements_qui_verrouillent = []

Pour chaque mouvement:
    nouvelle_position = effectue mouvement depuis position
    nouv_chemin = chemin prolongé de mouvement
    Si nouvelle_position est valide:
        ajoute (nouvelle_position, nouv_chemin) à à_visiter
    Sinon:
        ajoute mouvement à mouvements_qui_verrouillent

Si mouvements_qui_verrouillent est non vide:
    mouvement = mouvements_qui_verrouillent[0]
    Si position suffisamment basse:
        ajoute (position, chemin, mouvement) à verrouillables
```

En utilisant une file plutôt qu'une pile on obtient un parcours en largeur et on a donc la garantie que les chemins obtenus soient de longueur minimale. Cela n'a pour le moment pas d'importance mais il nous a semblé plus naturel de procéder ainsi au début du concours car les pièces mettaient beaucoup de mouvements à venir en position car elle zigzaguaient de gauche à droite.

Plus tard, nous nous sommes servi de la propriété du parcours en largeur pour intégrer les phrases spéciales dans l'algorithme.

## 4 Optimisation des paramètres par algorithme génétique

Afin de sélectionner la meilleure feuille de l'arbre construit par l'algorithme principal, il faut lui attribuer un score pondéré par un certain nombre de paramètres. Ces coefficients de pondération ont été choisis « au doigt mouillé » dans un premier temps en fonction de l'intuition que l'on avait de leur effet potentiel, mais une fois l'algorithme lâché sur un problème, il n'est pas certain que le choix soit optimal. On a donc décidé d'utiliser un algorithme génétique pour essayer de trouver rapidement un jeu de pondérations qui puissent faire mieux que le jeu par défaut.

### 4.a Idée de l'algorithme génétique

Le principe de l'algorithme génétique est de faire « s'affronter » différents jeux de paramètres pour pouvoir les classer en fonction d'un certain critère (ici, ce sera le score total sur l'ensemble des problèmes soumis). Une fois les différents candidats rangés par ordre d'efficacité, on sélectionne les meilleurs et on les « reproduit » entre eux en mélangeant leurs caractéristiques principales pour former un certain nombre de programmes « fils ». Les parents et les enfants s'affrontent alors à nouveau sur le jeu de problèmes et on resélectionne les meilleurs du cheptel pour la reproduction afin de produire une nouvelle génération. Au bout d'un certain nombre de générations (les effets commencent à se faire sentir à partir de la troisième), la « sélection naturelle » fait ressortir des jeux de paramètres qui peuvent notablement améliorer le score du jeu initial.

## 4.b Sélection initiale des candidats

Le génome initial des candidat est choisi aléatoirement en prenant une valeur entre 0 et 10 pour chacun des 6 paramètres que l'on doit fournir au programme principal. On prend tout de même un candidat correspondant au jeu de paramètres par défaut pour avoir un élément de comparaison et pouvoir décider si le jeu de paramètres trouvés par l'algorithme est meilleur ou non que le jeu initial.

```
import random as rd
ANCETRE = [3,2,1,100,1,100] # La pondération "naturelle" de base
CANDIDATS = [ANCETRE]      # Les candidats "préqualifiés"
NOMBRE_DE_CANDIDATS= 12    # Pour une génération
MAX_VAL = 10               # Pour un paramètre
NB_PARAMETRES = len(ANCETRE) # Le nombre de paramètres

def cree_candidat():
    """ Création d'un candidat aléatoire. """
    return [rd.randint(0,MAX_VAL) for j in range(NB_PARAMETRES)]

for i in range(NOMBRE_DE_CANDIDATS-len(CANDIDATS)):
    CANDIDATS.append(cree_candidat())
```

## 4.c Organisation du tournoi

Le tournoi en lui-même consiste simplement à appeler l'exécutable Caml sur les problèmes voulus avec les paramètres portés par chaque candidat. On récupère le score obtenu sur chaque problème de sorte à pouvoir classer les candidats en fonction de leur score total sur l'ensemble des problèmes que l'on choisit de tester afin d'obtenir un jeu de paramètre qui réussisse au mieux dans la plupart des cas qui se présentent. Afin de conserver une trace écrite, on rentre les résultats pour chaque problème dans la base de données préalablement initialisée.

## 4.d Reproduction des meilleurs

Après sélection des 4 meilleurs de la génération en cours, on les « reproduit » deux à deux pour essayer de trouver les « gènes » qui ont permis cette réussite. On pondère le mélange de gènes en fonction de la réussite au tournoi précédent de sorte à favoriser légèrement le plus fort. Néanmoins, on introduit un brin d'aléatoire de deux manières différentes :

1. 9 fois sur 10, on fait la moyenne pondérée des gènes des deux reproducteurs avec le poids précédent auquel on rajoute un facteur (positif ou négatif) aléatoire pour s'assurer que tous les gènes ne sont pas mélangés toujours dans les mêmes proportions<sup>5</sup>.
2. 1 fois sur 10, une mutation majeure intervient et les gènes parentaux sont tout bonnement ignorés. Le gène de l'enfant est alors tiré de manière aléatoire sur l'ensemble des valeurs initialement prévues.

---

5. Comme Harry Potter, on peut très bien hériter des yeux de sa mère et pourtant avoir globalement le même visage que son père.

```
def reproduction(p1,r1,p2,r2):
    """ Mélange des gènes pour p1 et p2. Renvoie un enfant.
    r1 et r2 sont les résultats respectivement pour p1 et p2. """
    poids = r1/(r1+r2)
    enfant = []
    for i in range(NB_PARAMETRES):
        # 9 fois sur 10, c'est le mixage normal
        if rd.randint(0,9) < 9:
            # On met un peu d'aléatoire dans le mélange
            coeff = poids + (-1)**rd.randint(0,1) * rd.random()
            enfant.append(int(coeff*p1[i] + (1-coeff)*p2[i]))
        else: # Sinon, mutation aléatoire
            enfant.append(rd.randint(0,MAX_VAL))
    return enfant
```

#### 4.e Importance des mutations et du sang neuf

Tout comme dans la vraie vie, le consanguinisme guette dès que l'on procède à la reproduction des candidats. C'est d'autant plus vrai ici que le programme Caml impose aux paramètres d'être des entiers. Même avec un soupçon d'aléatoire dans le brassage, mélanger deux fois la même valeur pour un paramètre donné redonne la même valeur. Si cette valeur fournit effectivement un avantage dans le tournoi, les meilleurs places seront brigués par les mêmes candidats. Il n'y aura plus d'évolution qui pourrait permettre de trouver un jeu de paramètre inédit et qui amènerait pourtant de meilleurs résultats. C'est pourquoi les mutations sont particulièrement importantes et leur pourcentage d'apparition (ici 10% de chance) est un paramètre sur lequel on peut jouer pour optimiser l'algorithme génétique lui-même.

Néanmoins, ces mutations peuvent ne pas être suffisantes pour explorer efficacement l'espace des paramètres, c'est pourquoi on a choisi de prendre 12 candidats pour chaque génération :

- Quatre d'entre eux sont issus de la génération précédente et remettent simplement leur titre en jeu <sup>6</sup>.
- Les six suivants sont les candidats « fils » qui correspondent aux six appariements deux à deux des quatre candidats précédents.
- Enfin, les deux places restantes sont constituées d'individus au génome totalement nouveau et choisi aléatoirement pour permettre d'explorer d'autres possibilités.

L'ensemble permet assez rapidement de trouver de nouveaux arrangements qui améliorent sensiblement les choix initiaux, les points forts de chaque individus étant patiemment sélectionnés pour se développer lors des générations ultérieures.

6. Bien entendu, comme il n'y a pas d'aléatoire dans le problème et que chaque joueur joue absolument seul, il ne sert à rien de faire tourner à nouveau les jeux de paramètres des parents dans le nouveau tournoi vu qu'ils donneront exactement le même résultat qu'avant (ce ne serait pas le cas par exemple dans un jeu où l'on affronte un autre joueur en combat singulier et que notre stratégie peut être dépendante des mouvements de l'adversaire). C'est pourquoi les résultats ont été stockés dans une base de données `sqlite3` et seuls les calculs sur les candidats fils ainsi que les nouveaux candidats aléatoires sont utilisés.

## **5 Conclusion**

Venez essayer vous aussi à l'adresse <http://icfp15.de-falco.fr/>!