13 avril 2018

Projet COREWAR

Travail Personnel Approfondi

L2 Informatique

Groupe 1A

Année 2017-2018

Table des matières

Introduction

Pour les étudiants en informatique, la programmation orientée objet apparaît comme un passage obligatoire. Ainsi, afin d'en appliquer les principes et de parfaire notre maîtrise de Java, des projets nous ont été proposés dans le cadre de l'unité d'enseignement de TPA (Travail Personnel Approfondi).

Par groupe de 4, il nous a été demandé de réaliser des applications complètes, sur différents thèmes : jeux vidéos, sténographie, etc.

Parmi ces propositions, si plusieurs ont retenu notre intention, une, plus que les autres, a capté notre intérêt : Le Projet Corewar, dont l'énoncé est le suivant :

> « Le but de ce projet est fournir un outil permettant de générer des programmes efficace au CoreWar. CoreWar est un jeu de programmation dans lequel deux programmes informatiques sont en concurrence pour le contrôle d'une machine virtuelle appelée MARS (Memory Array Redcode Simulator). Le but du jeu est de faire se terminer toutes les instances du (ou des) programme(s) adverse(s). Dans un premier temps, il faudra développer une plateforme de simulation de la machine virtuelle (en utilisant une version simple du langage de programmation appelé RedCode). Dans un second temps, il s'agira d'être capable d'exécuter les programmes écrits en RedCode et de déterminer le vainqueur. Dans la dernière étape, il faudra proposer une méthode (construction aléatoire, système à base de règles ou encore algorithme génétique) permettant d'obtenir un programme performant. »

Ici, ce projet est décomposé en 3 parties :

- Le développement d'une plateforme de simulation de la machine virtuelle où s'affronteront nos warriors ;
- La conception d'un interpréteur afin d'être en mesure d'exécuter les programmes en Redcode ;
- La création d'un générateur de programmes.

1 Redcode et Implémentation

1.1 Corewar

Corewar est un jeu de programmation dont la première version officielle est apparue en 1984, à l'initiative de D. G. Jones et de A. K. Dewdney. [9]

Comme expliqué dans le sujet, le principe du jeu est plutôt simple : deux (ou plus) programmes (dénommés « Warriors »), codés en Redcode, et disposés aléatoirement sur la mémoire cyclique d'une machine virtuelle, s'affrontent à coup d'instances ; le but étant de mettre fin à celles de l'adversaire, pour prendre le contrôle de la machine.

Au sujet du Redcode

Le Redcode est un langage assembleur relativement basique. Chaque programme est formé d'une ou plusieurs lignes composée(s) de la façon suivante :

FIGURE 1 – La composition d'une ligne de commande en Redcode

<label> : ▢ <opecode> ▢ <A-mode> <A-field> ▢ <B-mode> <B-field>

L'ensemble "<A-mode><A-field>" (ou "<B-mode><B-field>") est appelé « operand ».

Labels Les labels sont des constantes symboliques qui peuvent être appelées sans passer par un calcul d'adresse.

Opecode Les opecodes, ou "operateurs" sont des operations de base qui constituent les Warriors, c'est à dire les programmes qui s'affrontent dans le CoreWar. (voir 2 page 5)

Modes d'adressage Les modes d'adressages, contenus dans <A-mode> et <B-mode>, sont des symboles qui permettent de définir la façon dont sera/seront calculé(s) la/les adresse(s) cible(s) pour l'opérateur associé.

Modifieurs Les modifieurs permettent de définir quel Field (A ou B) sera affecté par l'operateur associé.

P-Space Les P-Spaces sont des espaces réservés à un joueur.

1.2 Versions

Le Redcode, du haut de ses 30 et quelques années, a subit de nombreuses modifications et mises à jour depuis sa version initiale. Originellement composé de seulement 8 instructions, ce langage assembleur s'est largement développé en plusieurs versions (ICWS [1] 86, ICWS 88, etc), dont certaines non officielles, implémentant divers fonctionnalités, ainsi que nombreuses autres instructions

TABLE 1 – Comparatif des differents versions du RedCode

	Base (1984)	ICWS'86	ICWS'88	ICWS'94 Standard Hill	Version Commune	Bloup
Nombre d'opecode	8	11	13	17	19	17
Modes d'adressage	3	4	3	5	8	8
Modifieurs	x	x	6	7	7	x
P-Space	x	x	x	x	v	x
Labels	x	v	v	v	v	x
Commentaires	x	;	;	;	;	x

La version BLOUP [2]

Etant donné le nombre de versions existantes et par soucis de tenir les délais pour ce projet, nous avons décidé d'implémenter notre propre version répondant au nom de BLOUP. La version BLOUP dispose de caractéristiques communes avec les versions précédentes du Redcode ; mais elle possède aussi ses propres spécificités [3].

1. ICWS signifie «International Core Wars Society»[17]
2. BLOUP signifie «Binary list organized usurpation programming», autrement dit, pas grand chose
3. basées sur les informations du «Beginners' guide to Redcode»[13]

TABLE 2 – Tableau des Caracteristiques de la version Bloup du RedCode

Capacités Communes	Opecode	**DAT** : Tue le processus
		MOV : Copie les datas d'une adresse à une autre
		ADD : Ajoute un nombre à un autre
		SUB : Soustrait un nombre à un autre
		MUL : Multiplie un nombre par un autre
		DIV : Divise un nombre par un autre
		MOD : Donne le reste de la div. d'un nombre par un autre
		JMP : Reprend l'éxécution à une autre adresse
		JMZ : Fais un jump si la data testée est 0
		JMN : Fais un jump si la data testée n'est pas 0
		DJN : Decrémente la data testée et fais un jump si c'est 0
		SPL : Lance un nouveau processus à une nouvelle adresse
		CMP : Compare deux data, saute l'instruction suivante si elles sont egales
		SEQ : Comme CMP
		SNE : Compare deux data, saute l'instru. suivante si elles ne sont pas egales
		SLT : Compare deux data, saute l'instru. suivante si Data1 $<$ Data2
		NOP : Ne fais rien
	Adressage	# : Immédiat (Valeur)
		$: Direct (le $ n'est pas obligatoire, il s'agit de l'adressage par defaut)
		***** : Field A Indirect
		@ : Field B Indirect
		} : Field A Indirect post-incrémenté
		{ : Field A Indirect pré-incrémenté
		$>$: Field B Indirect post-incrémenté
		$<$: Field B Indirect pré-incrémenté
Spécificités de BLOUP	Une adresse indirecte doit pointer sur un mode immédiat.	
	Nos Warriors sont des fichiers ".bloup". Crée pour l'occasion, cette extension nous permet de faire la différence avec des Warriors complexes trouvables sur internet que notre interpreteur n'est pas capable de lire.	
	Le nombre de cases memoire minimal entre 2 programmes sur MARS est de 100	

2 Organisation du projet

2.1 Répartition des tâches

Le projet étant composé de trois objectifs clairement distincts (voir page 2), nous avons d'abord travaillé tous ensemble à l'élaboration d'un plan de travail, et ce avant même de commencer à coder. Cette session de reflexion nous à d'abord permit de cerner les deux premiers objectifs du projet : La mémoire et l'interpréteur. Afin d'optimiser le temps de travail, nous nous sommes dans un premier temps séparés en binômes. Corentin et Marion se sont concentrés sur la Mémoire, tandis que Florian et Morgane ont actés à la réalisation de l'interpréteur.

Malgré le travail préparatoire, la conception de ce dernier à demandé bien plus de temps que prévu, notamment du fait du calcul d'adresse (voir 3.1 page 9). Ainsi, la Mémoire étant terminée bien plus rapidement que l'interpréteur, Marion et Corentin ont pu commencer des recherches sur les algorithmes génétiques, et, plus généralement, sur la création d'un générateur de programmes (relativement) performants.

Les tests de l'interpreteur n'étant d'abord pas particulièrement fructueux, il nous à fallut concevoir des warriors adaptés à notre version du RedCode. Cela nous à permit de soulever des erreurs sur lesquelles nous avons travaillé tous ensemble, en parralèle de la conception du générateur.

2.2 Architecture du programme

Concernant l'architecture du programme, nous avons voulu pleinement coller au sujet. Nous avons donc originellement opté pour une décomposition en 3 packages :

GestionMemoire qui, comme son nom l'indique, permet de gérer la mémoire MARS [4], et son contenu.

Interpretation le package qui permet d'interpreter le Redcode

AlgoGen package propre à la géneration de programmes

Après un certain temps dans cette configuration, nous avons pris conscience que de nombreux attributs étaient redondants à la totalité de notre code. Nous avons donc décidé des les stocker dans une class final (qui ne peut donc pas être dérivées). Les attributs, ici Static, sont donc accessibles en dehors de la class sans pour autant avoir à la construire.

4. Mars signifie «Memory Array Redcode Simulator»

FIGURE 2 – Diagramme de la classe GestionMemoire

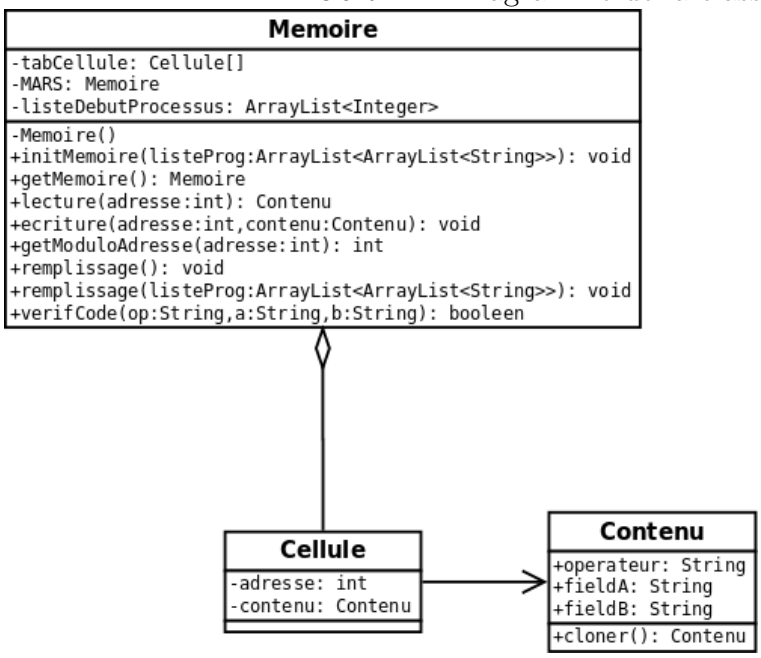


FIGURE 3 – Diagramme de la classe Interpretation

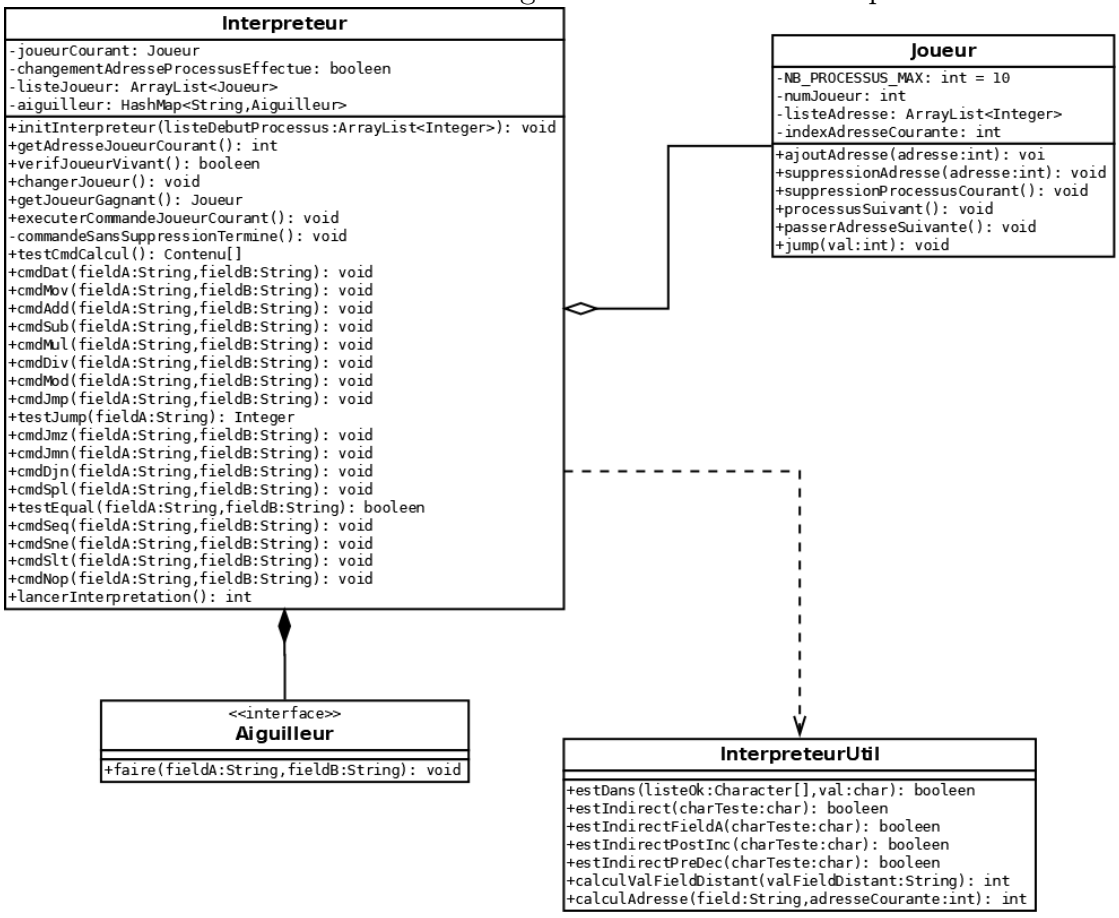


FIGURE 4 – Diagramme de la classe AlgoGen

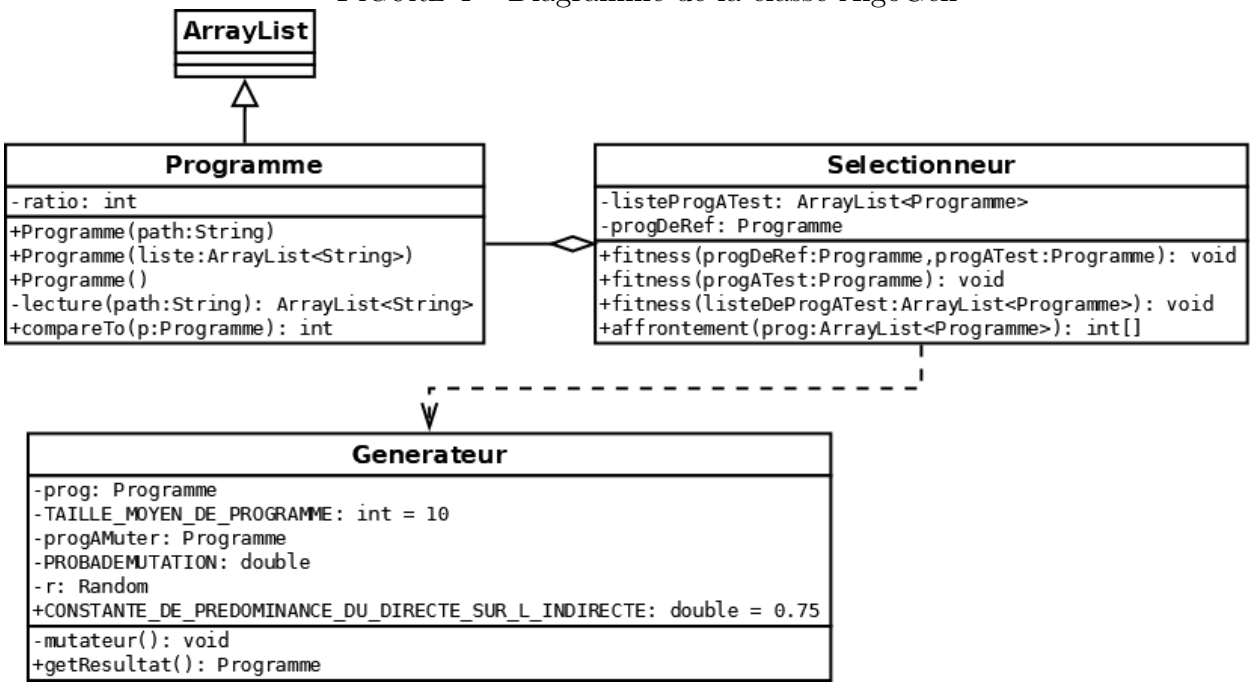
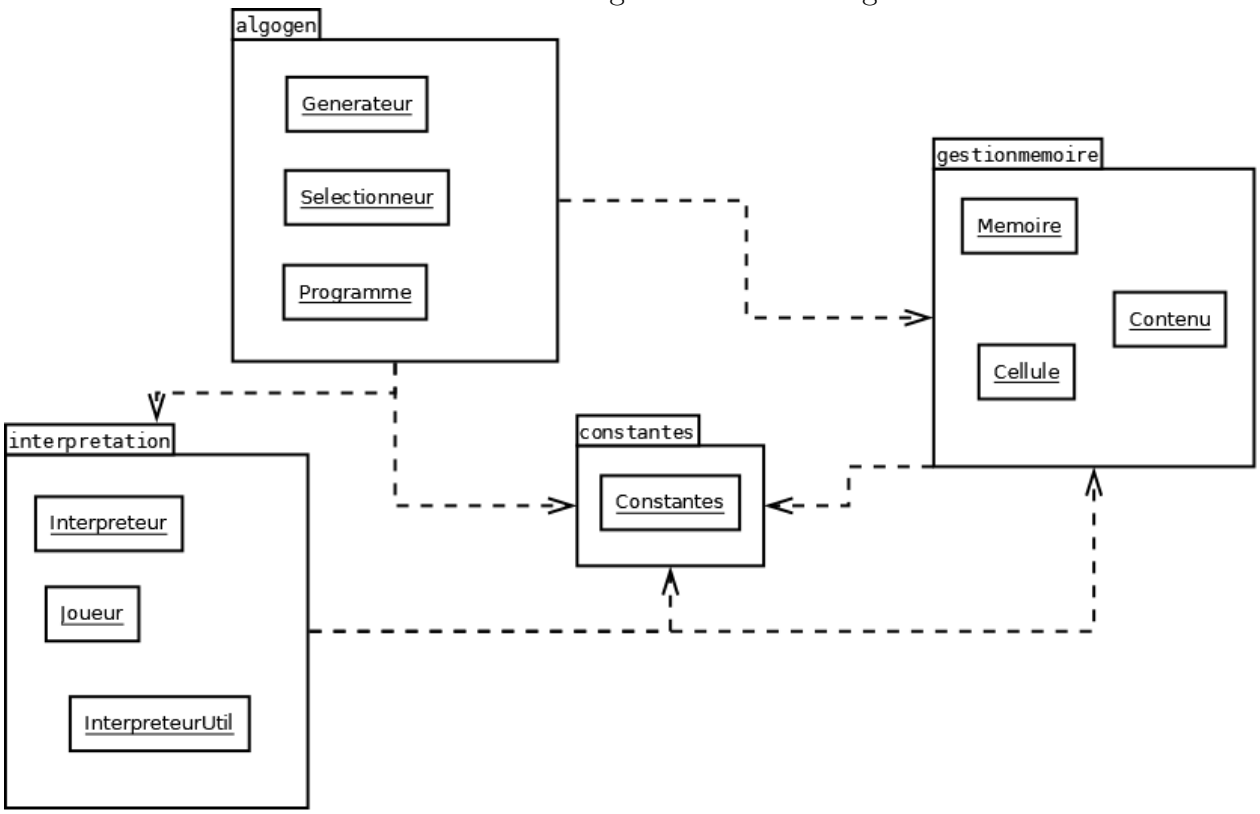


FIGURE 5 – Diagramme des Packages



3 Elements techniques

3.1 Interpreteur

La méthode CalculAdresse

La méthode calculAdresse a pour but de connaître l'adresse pointée par une adresse directe ou indirecte. Elle prend en argument un « operand » (nommé field au sein du code) qui peut être décomposé en deux parties (comme montré dans la figure 1, page 3) :

- un «mode» :
  - «immédiat» (#) : une simple donnée
  - «direct» ($ ou rien)
  - «indirect» (({,},<,>,*,@)
- une valeur

Si le field est un immédiat, la méthode retourne -1 pour l'indiquer au programme qui a appelé la methode. (Le -1 sera alors interprété plus tard)

Une adresse indirecte a pour but d'aller chercher un immédiat dans une cellule distante (du moins, dans notre version, cf 1.2) et d'ajouter à son adresse la valeur contenue dans un de ses fields (dans le cas où le field ciblé ne contient pas un immédiat on retourne -2 pour indiquer que le calcul indirect est invalide) et on retourne cette valeur.

Dans le cas d'une adresse directe on retourne simplement l'adresse qui est a n case mémoire de la case courante n étant la valeur du field.

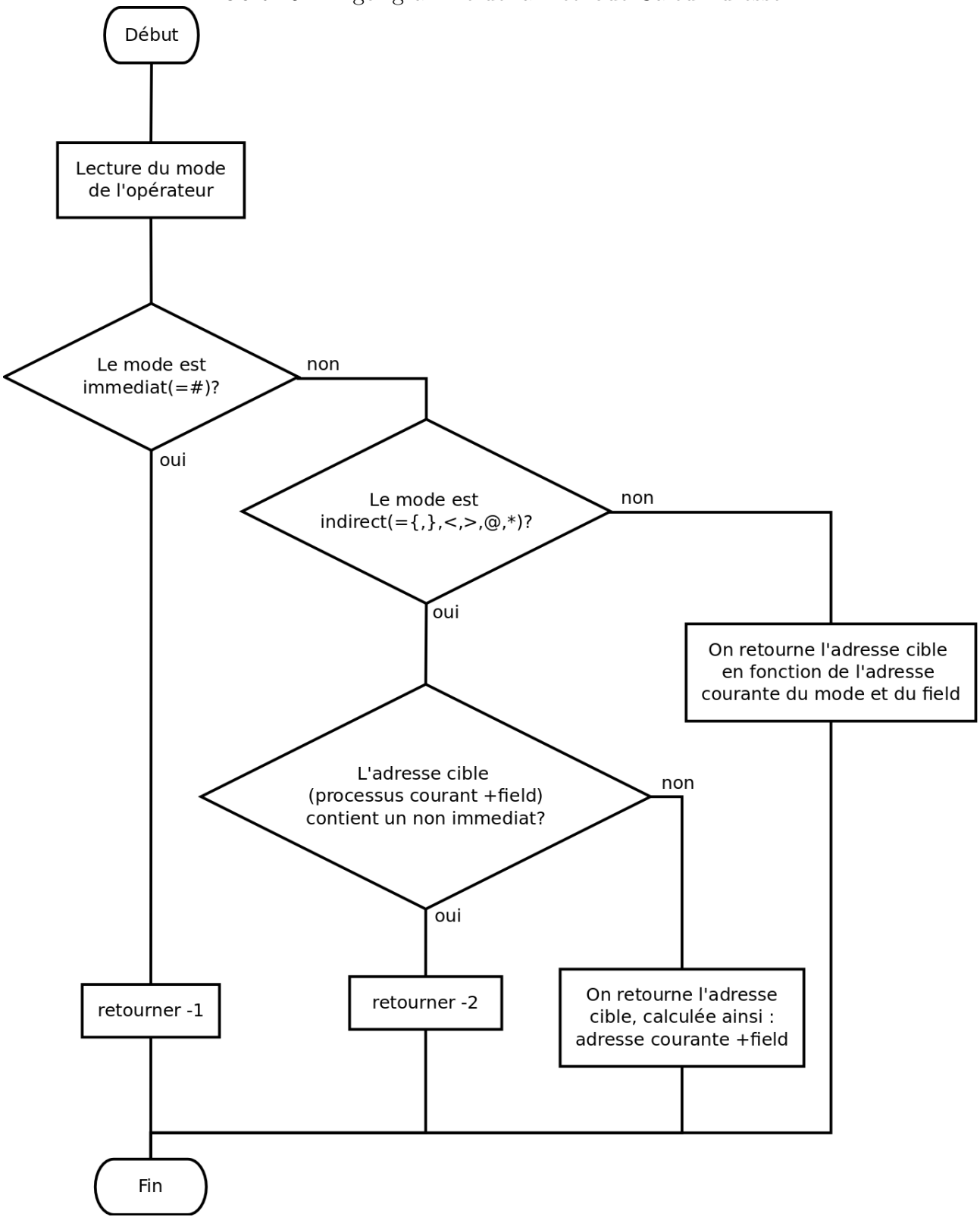
Gestion processus

Le package interpretation permet d'interpréter des commandes RedCode stockées dans une mémoire. L'*Interpreteur* va récupérer le *Contenu* d'une *Cellule* et va ensuite, selon l'opérateur, appeler la commande associé qui « exécutera la commande ». Chaque programme est associé à un *Joueur*, un affrontement se fait au tour par tour et de ce fait l'Interpreteur exécute les commandes des joueurs avec le même procédé. En début de partie les joueurs ont tous un processus qui correspond a l'adresse de la *Cellule* contenant la première instruction de leur programme. Au cours d'une partie, les joueurs peuvent créer de nouveaux processus (10 au maximum), ils seront alors exécutés les uns après les autres (cf. figure 7 page 11).

3.2 Memoire

Le package gestionmemoire a pour but de simuler une machine virtuelle (MARS[2]) qui permettra de stocker les programmes Redcode qui pourront ensuite être exécutés. MARS est cyclique et contient un ensemble de *Cellule* qui ont chacune une adresse associée, les adresses se suivent et vont de 0 à 'la taille de la mémoire -1'.

FIGURE 6 – Algorigramme de la methode CalculAdresse



Design Pattern Singleton

— Qu'est ce qu'une classe Singleton ?

Il s'agit d'une classe qui ne peut être instanciée qu'une seule fois.

— Quelle utilité pour notre programme ?

FIGURE 7 – Représentation des processus

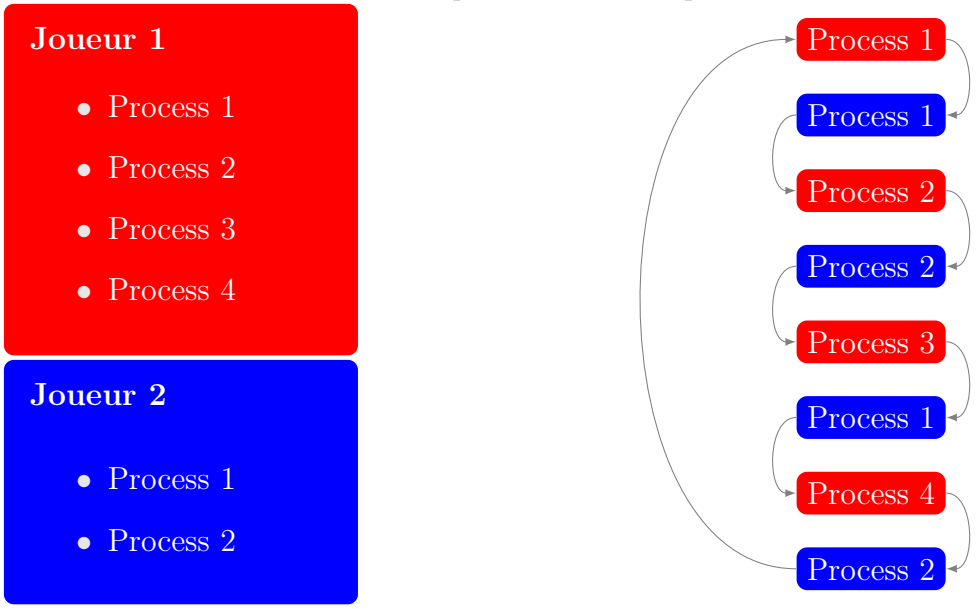
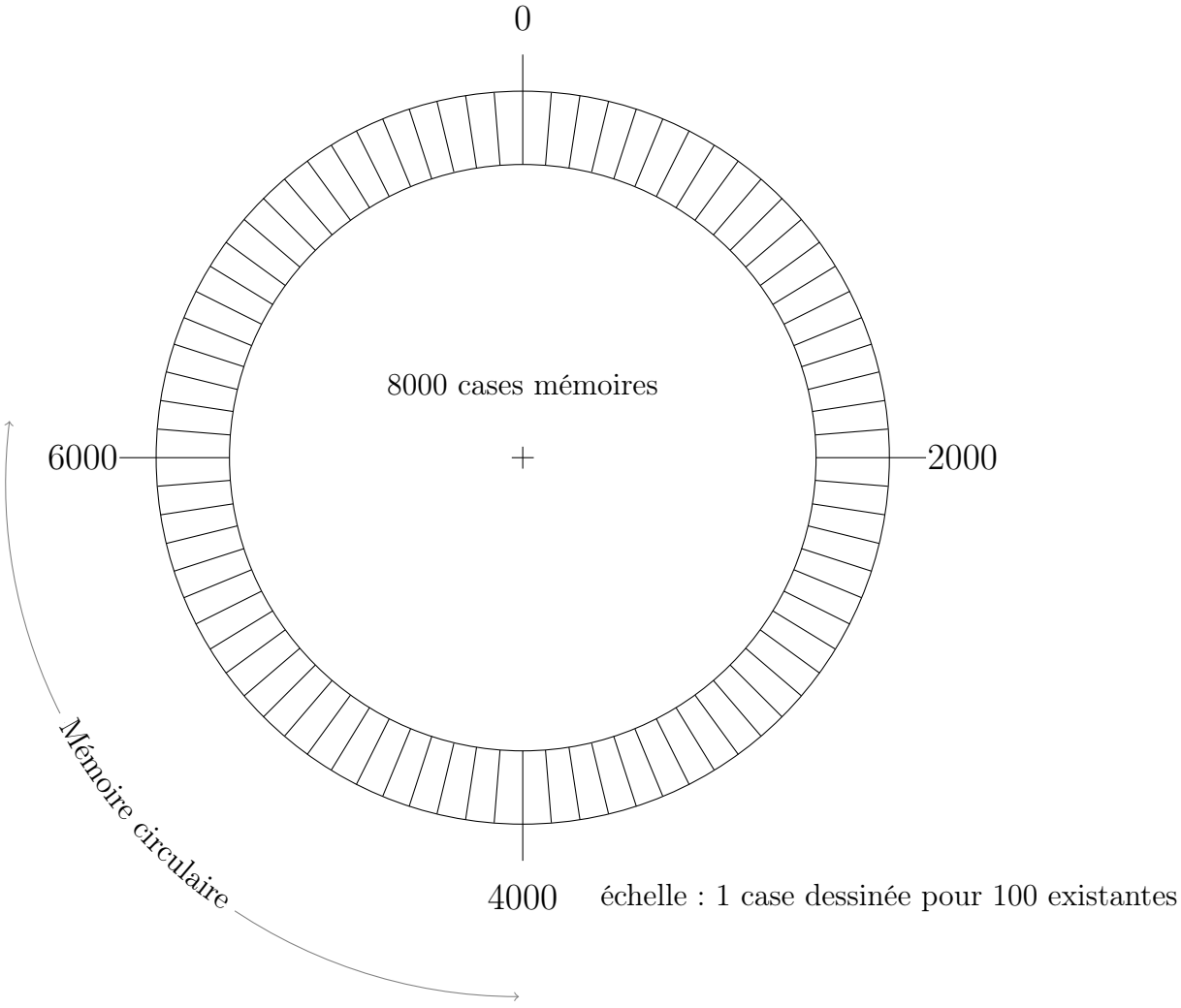


FIGURE 8 – Représentation de la Memoire



Dans le cadre de notre projet il s'agit d'éviter les conflits lors de l'exécution d'une partie ainsi la mémoire ne peut être dupliquée et une unique partie peut être jouée.

Lecture fichier

Le stream est une façon efficace de parcourir une source de données en java. Ici il permet de parcourir un dossier et récupérer les paths de tous les fichiers d'un format particulier (ici .bloup) qui seront donc interprétable. Ensuite, avec un for each on recupère les paths de chaque fichier. Enfin on stock toutes les lignes des fichiers dans une liste de Strings ce qui nous permet par la suite de traiter ces lignes de manière simplifiée.

FIGURE 9 – Utilisation d'un Stream

```
Path emplacement = Paths.get("../dist/redCode/exemple");
try (DirectoryStream<Path> stream = Files.newDirectoryStream(emplacement,"*.{bloup}")){
        for (Path entry : stream) {
        List<String> lignes =Files.readAllLines(FileSystems.getDefault().getPath(entry.toString()));
```

Ecriture memoire

Algorithm 1 écriture en mémoire

```
function REMPLISSAGE(listeProg)
    for Fichiers .bloup to ../dist/redCode/exemple do
        for Lignes to Fichiers do
            if Lignes respecte la nomenclature de notre version Bloup then
                l ← Lignes
            end if
        end for
        adresse ← un nombre aléatoire (distance entre prog =100 au minimum)
        ligne par ligne la liste l contenant toutes les lignes du code respectant la nomenclature
    end for
end function
```

3.3 Génération de programme

En ce qui concerne la génération de programmes, nous avons conçu deux façons de procéder :

- La première que nous ayons mise en place est la mutation[14]. Il s'agit d'une technique qui se base sur un programme préexistant. Ce programme est parcouru champ après champ ; chaque champ ayant une certaine chance d'être muté.

$$Proba = \frac{x}{tailleProg * 3} \; x \in]0, 6]$$

Si le champ qui mute est un opérateur, le mutateur en sélectionne un, aléatoirement, dans la liste de tous les opérateurs que notre version du redCode peut interpreter. En revanche, si le champ qui mute est un field, alors il sélectionne un prédicat direct/indirect qui a 75 % de chance d'être un indirect, suivi d'un nombre qui va de 'moins la taille du programme' (pour que nos opérateurs ne s'appliquent pas sur des DAT automatiquement), jusqu'à '10-la taille du programme' (Car un programme fait en moyenne 10 lignes, cela évite ainsi que certaines lignes ne s'appliquent en dehors du programme [5]). Nous aurions aussi pu ajouter le fait de complètement supprimer une ligne du programme qui nous sert de base (et donc, de ne pas la muter) ou d'en écrire une qui ne soit pas du tout dans ce programme.

- La deuxième méthode pour générer un programme que nous ayons mise en place est, justement, de générer un programme à partir de rien. Ainsi, sur plus ou moins 10 lignes, un opérateur est choisi aléatoirement, suivi des fields qui se construisent comme précédement : un prédicat, qui a 75 % de chance d'être indirect, puis un nombre qui va de 'moins la taille du programme' jusqu'à '10-la taille du programme'.

5. Certains programmes construits le font sans risque car ils sont réfléchis

Conclusion

Objectifs remplis ?

Au cours de la réalisation de ce projet, nous avons été à de nombreuses reprises pris par le temps. Bercés par de grandes ambitions (gestion de la totalité des opecodes, de tous les modes d'adressage) nous n'avons pas pu pleinement aboutir.

Malgré cela, notre mémoire est parfaitement fonctionnelle. À son tour, l'interpreteur ne dispose certes pas d'un nombre saisissant de fonctionnalités, mais celles ayant été implémentées ne présentent aucunes difficultés d'exécution.

Par ailleurs, nous ne sommes pas pleinement satisfaits de notre générateur. Le problème étant qu'un bon warrior ne se définit pas en fonction de son nombre d'instructions ou de leur nature, mais de la façon dont celles ci sont combinées. Aléatoirement, obtenir des résultats pertinents est donc très rare. L'algorithme génétique annihile d'une façon ou d'une autre les combinaisons d'instructions qui font leur force, et encore une fois, cela nuit évidement à la qualité du programme. [6]

Améliorations possibles

Il apparait comme évident que de nombreuses améliorations sont possibles pour un tel projet. En premier lieu, nous pourrions implémenter plus de caractéristiques du Redcode. Les P-spaces dans la mémoire que nous avions entamés pourraient être mis en place. Ces derniers n'ont pas pu aboutir par manque de temps.

De la même façon, l'implémentation des commentaires et des labels dans les fichiers redcode nous aurait permis d'utiliser des fichiers trouvables sur internet sans avoir à les retoucher au préalable.

Concernant le générateur, nous pourrions essayer d'autres façons de procéder pour l'algorithme génétique en pratiquant par exemple des croisements afin d'obtenir des warriors plus performants.

Finalement, une interface graphique aurait constitué un apport intéressant à un tel programme.

Malgré tout, nous retiendrons de grandes leçons : la necessité d'une bonne préparation, l'importance des recherches, la valeur du temps dans la réalisation d'un programme, etc. Outre la complexité de ce projet auquel nous nous sommes attelés par curiosité, nous avons découvert bien plus qu'un jeu regroupant des passionnés : les Warriors représentent à la perfection le fonctionnement des virus (exécution de processus, occupation de l'espace libre, etc), et cela offre évidemment nombre de perspectives informatiques.

6. On peut tout de même dire que notre générateur produit des programmes performants, si « performant » signifie « ne pas se suicider au premier tour ».

Références

[1] Redcode reference (icws'94 draft (extended)) [en ligne]. Disponible sur http://www.koth.org/info/pmars-redcode-94.txt, 2000.

[2] Koth.org [en ligne]. Disponible sur http://www.koth.org/index.html, 2010.

[3] A brief history of corewar [en ligne]. Disponible sur http://corewar.co.uk/history.htm, 2016.

[4] Corewar.info [en ligne]. Disponible sur http://www.corewar.info/, 2016.

[5] Wikibooks, core war/redcode [en ligne]. Disponible sur https://en.wikibooks.org/wiki/Core_War/Redcode, 2017.

[6] Developpez.com [en ligne]. Disponible sur https://www.developpez.net/forums/, 2018.

[7] Openclassrooms [en ligne]. Disponible sur https://openclassrooms.com, 2018.

[8] Stackoverflow [en ligne]. Disponible sur https://stackoverflow.com/, 2018.

[9] Wikipedia, core war [en ligne]. Disponible sur https://en.wikipedia.org/wiki/Core_War, 2018.

[10] Stephen Beitzel and Mark Durham. Annotated draft of the proposed 1994 core war standard [en ligne]. Disponible sur http://www.corewars.org/docs/94spec.html, 1995.

[11] Sapan Bhatia. Corewars [en ligne]. Disponible sur http://www.corewars.org/index.html, 2011.

[12] D. G. Jones and A. K. Dewdney. Core war guidelines [en ligne]. Disponible sur https://users.obs.carnegiescience.edu/birk/COREWAR/DOCS/guide2red.txt, Mars 1984.

[13] Ilmari Karonen. The beginners' guide to redcode [en ligne]. Disponible sur http://vyznev.net/corewar/guide.html, 2012.

[14] khayyam90. Les algorithmes genetiques [en ligne]. Disponible sur https://khayyam.developpez.com/articles/algo/genetic/, 2005.

[15] Terry Newton. Introduction to redcode [en ligne]. Disponible sur http://www.infionline.net/~wtnewton/corewar/tutor.txt, 1997.

[16] John Perry. Core wars genetics [en ligne]. Disponible sur http://corewar.co.uk/perry.htm, 2012.

[17] Lawrence Woodman. An introduction to corewar [en ligne]. Disponible sur http://techtinkering.com/2009/04/30/an-introduction-to-corewar/, 2009.