Corewar 3

1. Qu’est-ce que le Corewar ?

Deux programmes minimum, respectant les règles d'un jeu, vont s'affronter dans la mémoire vive d'un ordinateur et vont s'attaquer mutuellement jusqu'à ce qu'il n'y ait qu'un survivant.

Les programmes sont chargés dans des cellules quelconques d'un espace mémoire d'une machine virtuelle et l'attaquant, qui ignore où sont situés les segments de son adversaire, doit le localiser puis le détruire. A tour de rôle, un programme attaque (il peut tomber sur des cellules vides) puis c'est à lui d'attaquer etc...

2. Notre version du Redcode :

Les programmes (appelés « guerriers ») sont écrits au préalable dans un langage assembleur appelé Redcode. La première version du Redcode proposait 8 instructions différentes sur une architecture 32bits. La technologie avançant, de nouvelles instructions ont été ajoutées à ce langage et le jeu est passé en 64bits. Deux versions iconiques du Redcode ont été retenues, une créée en 1984 (8 instructions basées sur une architecture 32bits) et une en 1994 (plus d’instructions et de modes d’adressage basés sur une architecture 64bits).

La version utilisée dans notre programme est un mélange de ces deux versions. Nous voulions garder l’architecture 32bits tout en ajoutant quelques instructions afin d’augmenter les opérations possibles des joueurs.

Voici les 16 instructions qui ont été implémentées :

Encodage	Symbole Mnémonique	Argument	Argument	Sémantique
0	DAT	A		Initialise à la valeur A l’adresse actuelle.
1	MOV	A	B	Se copie en se déplaçant de B cases en partant de A.
2	ADD	A	B	Ajoute l’opérant A au contenu de l’endroit B et stocke le résultat à l’endroit B

3	SUB	A	B	Soustrait l'opérant A au contenu de l'endroit B et stocke le résultat à l'endroit B.
4	JMP	A		Continuer à l'adresse B.
5	JMZ	A	B	Si l'opérant A est 0, se déplace à l'endroit B ; sinon continue avec l'instruction suivante.
6	DJZ	A	B	Décrémente le contenu de l'endroit A de 1. Si l'endroit A dispose maintenant d'une valeur de 0, se déplace à l'endroit B ; sinon continue avec l'instruction suivante.
7	CMP	A	B	Compare l'opérant A avec l'opérant B. Si ils ne sont pas égaux, saute l'instruction qui suit ; sinon continue avec l'instruction suivante.
8	MUL	A	B	Multiplie l'opérant A au contenu de l'endroit B et stocke le résultat à l'endroit B.
9	DIV	A	B	Divise l'opérant A au contenu de l'endroit B et stocke le résultat à l'endroit B.
10	MOD	A	B	Modulo
11	NOP			Aucune opération.
12	JMN	A	B	Saute si non 0 (teste une instruction et saute si elle n'est pas égale à 0)
13	DJN	A	B	Décrémente puis saute si non 0 (décrémente une instruction et saute si elle n'est pas égale à 0)
14	SEQ	A	B	Similaire à "CMP"
15	SNE	A	B	Saute si pas égal (compare deux endroits et saute l'instruction suivante si ils ne sont pas égaux)

Les différents modes d'adressage :

Codage	Symbole		Sémantique
0	<aucun>	Relatif	A : désigne l'adresse actuelle + A
1	#	Immédiat	#A : désigne l'adresse absolue A
2	@	Indirecte	@A : désigne l'adresse contenue à l'adresse actuelle + A

A noter : Il est interdit pour le mode d'adressage de la destination d'être absolu (#).

CAHIER DES CHARGES

3. Travail demandé (repris du document PDF mis en ligne sur moodle)

- Faire tourner des programmes en langage redcode d'origine.

- Organiser un duel de programmes. Les programmes redcode seront chargés depuis des fichiers texte. Les divers paramètres (nombre de tours...) seront paramétrables. On doit pouvoir comprendre ce qui se passe grâce à une trace (journal des instructions exécutées par la machine virtuelle) et un core dump : le contenu de la mémoire est écrit dans un fichier texte, si possible de façon lisible (sous forme d'instructions, en mode texte, et non binaire)

- Pouvoir visualiser la mémoire à chaque instant en colorant les cases selon le programme qui les a écrites.

- Proposer quelques programmes redcode originaux, c'est-à-dire qui ne se trouvent pas déjà sur Internet, sera apprécié.

- Il est intéressant d'implémenter également la version redcode94. Elle introduit en particulier la réplication (un processus-père engendre un fils) et l'espace mémoire privé. On pourra se contenter d'une version intermédiaire (par exemple redcode94 sans les instructions SPL LDP STP donc sans réplication ni mémoire privée) ou introduire de nouvelles instructions.

- Organiser des compétitions complexes, à la façon des Hills de http://www.koth.org (King of the Hill) ou selon toutes les règles que vous voudrez. En particulier des mécanismes darwiniens : au sein de la population, les programmes les plus compétitifs «se reproduisent», c'est-à-dire sont lancés en plus grand nombre lors du tournoi suivant, tandis que la population des moins compétitifs régresse.

4. Les différentes parties de l'interface graphique :

Tout notre programme a été créé en java/swing/awt. Notre interface graphique peut être séparée en plusieurs parties :

- La présence de 6 boutons :
    - « Joueur 1 » : Permet de charger un fichier texte dans lequel est écrit un guerrier en langage Redcode. Comme son nom l'indique, il s'agira du joueur 1.
    - « Joueur 2 » : Permet de charger un fichier texte dans lequel est écrit un guerrier en langage Redcode. Il s'agira ici du joueur 2.
    - « Lancer » : Ce bouton permet de lancer une partie. Si aucun joueur n'a été importé ou si un des joueurs est manquant, un code de base (appelé « imp ») sera sélectionné par défaut. Si un des joueurs est manquant et qu'une partie

a précédemment été lancée, le guerrier utilisé par ce joueur lors de la dernière partie sera sélectionné.
    - « Pause/Reprise » : Permet de mettre le jeu en pause puis de reprendre la partie.
    - « Arrêter » : Permet d'arrêter la partie. Le jeu sera alors remis à zéro.
    - « Vitesse » : 5 vitesses de déroulement de la partie à choisir.

- La grille de jeu (64x64) : Représentation graphique en temps réel du déroulement de la partie. Le joueur 1 est représenté en rouge et le joueur 2 en bleu.

- Les instructions effectuées par chaque joueur sont représentées en temps réel à droite de l'écran.

- La mémoire utilisée en temps réel dans la grille par chaque joueur.

- Les cycles restants avant la fin de la partie, un cycle se terminant lorsque chaque joueur a effectué une instruction. Si aucun des guerriers n'a vaincu l'autre, l'égalité sera déclarée entre les deux joueurs.

5. Le fonctionnement de notre code

    a) L'interprétation des instructions

Chaque case du core a une valeur de base de 0 quand elle est vide. Une valeur est affectée à chaque case après modification par un guerrier. Cette valeur est calculée comptes tenus du type d'instruction, des modes d'adressages et des valeurs de A et B de chaque instruction. Alors, si le joueur 1 modifie une case du processus du joueur 2, elle risque de ne plus être exécutable et terminera donc son processus, ce qui donnera la victoire au joueur 1.

    b) Le core

Le core est implémenté comme un tableau d'integers. Dans notre programme, le tableau dispose de 4096 (ou 2^12) éléments représenté graphiquement par un tableau de taille 64x64. Chaque élément du tableau a le format suivant :

Champs	type	Mode d'adressage de A	Mode d'adressage de B	A	B
Nombre de bits	4	2	2	12	12

Tout calcul concernant les adresses sont faits modulo 4096 (la taille du core). Par exemple, un déplacement de 1000 cases en partant de la case 4095 vous fera arriver à la case 999. A noter que les nombres négatifs sont gérés en partant du principe que -x = 4096-x. Cela signifie que les nombres sont interprétés comme positifs entre 0 et 2048 puis négatifs jusqu'à 4096. Chaque processus se déplace dans le core en interprétant les valeurs des cases comme expliquées dans la partie « L'interprétation des instructions ».