Bases de la programmation impérative

Ensimag 1A

1. Blobwar

On considère un jeu de stratégie : *blobwar*. Le jeu se joue à deux joueurs (les bleus et les rouges) sur un plateau de taille 8×8. Les cases sont numérotées de 0 à 63 (figure 1).

56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

FIGURE 1 – Numérotation des cases du plateau (*"positions"*)

Le plateau contient des pions (également appelés blobs) bleus et rouges. On joue chacun son tour ; à chaque tour le joueur courant choisit un de ses pions et le déplace. Tout pion peut se déplacer sur une case voisine vide (y compris en diagonale) en se dupliquant (créant ainsi un nouveau pion de même couleur). Un pion peut également se déplacer de deux cases ($\max(\Delta\text{ligne}, \Delta\text{colonne}) = 2$) en sautant, ce qui n'engendre pas de duplication. Une fois arrivé sur sa destination un pion transforme tous les pions voisins de son adversaire en pions de sa propre couleur. Le jeu s'arrête dès qu'un joueur ne peut pas jouer ; le joueur majoritaire remporte alors la partie.

La figure 2 illustre les règles du jeu (les blobs bleus sont représentés pleins et les blobs rouges, vides). Tout d'abord le joueur bleu duplique le pion de la position 49 sur la case 42. Ensuite, le joueur rouge fait sauter le pion de la case 24 sur la case 41, ce qui fait passer les trois pions voisins de bleu à rouge.

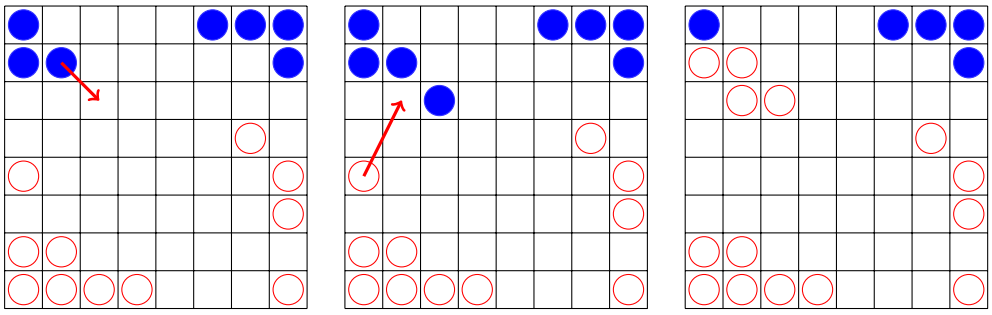

FIGURE 2 – Deux coups successifs

On définit la valeur d'un plateau comme étant le nombre de pions bleus moins le nombre de pions rouges.

Dans ce TD on regarde différentes solutions pour stocker les informations du jeu.

2. Tableau unidimensionnel

On représente dans un premier temps le plateau par un tableau `plateau` unidimensionnel de 64 cases correspondant aux 64 cases du plateau (les indices de `plateau` sont donc les *positions*). Chaque case du tableau contient 0 si la case correspondante du plateau est vide, et sinon 1 pour un blob bleu ou -1 pour un blob rouge.

2.1. Mettez un blob bleu ligne 3, colonne 2 (la case 0 étant ligne 0, colonne 0).

2.2. Écrivez une fonction renvoyant la valeur d'un plateau.

2.3. Écrivez un générateur `blobs` prenant en argument un plateau et une couleur (1 ou -1) permettant d'itérer sur toutes les positions des blobs de la couleur donnée.

2.4. Écrivez un générateur permettant d'itérer sur toutes les positions "voisines" d'une position donnée. On prendra comme paramètre supplémentaire la distance maximale autorisée.

2.5. Écrivez un générateur permettant d'itérer sur tous les coups possibles pour un joueur donné sur un plateau donné. Chaque coup est un couple (départ, arrivée).

2.6. Écrivez une fonction `jouer_coup(plateau, coup)` appliquant le coup donné sur le plateau.

3. 2 entiers

On se propose maintenant d'utiliser une représentation alternative du jeu. Comme le plateau ne contient que 64 cases, il est possible de stocker toute l'information sur les positions du joueur bleu à l'aide de 64 bits, ce qui correspond à un entier entre 0 et $2^{64} - 1$. Il nous suffit donc de deux entiers, un pour les bleus, un pour les rouges pour stocker une configuration du plateau. On utilisera pour ce faire un tableau `pions` composé de deux entiers. On identifiera maintenant les joueurs par les numéros 0 (bleu) et 1 (rouge).

3.1. Mettez un blob bleu case 9.

3.2. Enlevez un blob rouge de la case 12.

3.3. Écrivez un générateur permettant d'itérer sur toutes les positions d'un entier donné.

3.4. Écrivez une fonction renvoyant la valeur d'un plateau de jeu.

3.5. On se propose de coder pour une position donnée toutes les cases voisines (à distance 1) sous forme d'un entier de 64 bits. Proposez une fonction `calcul_voisins(position)` renvoyant l'entier en question. Par exemple `calcul_voisins(0)` doit renvoyer 770 (cases 1, 8 et 9).

3.6. On suppose maintenant disposer d'un tableau `voisins` précalculé, contenant pour chaque position l'entier codant ses voisins. Implémentez la fonction `jouer_saut(pions, coup, joueur, voisins)` modifiant l'état des blobs pour le coup donné effectué par le joueur donné, à l'aide du tableau voisins. Le coup donné est forcément un saut. Votre code ne devra utiliser aucune itération.