Worms

Marin Bougeret

Octobre 2011

Vous vous trouvez enseveli dans un tunnel. Vous allez peut être y passer. Cependant, vous vous souvenez de votre cours d'algo de 1ere année, et vous décidez de ne pas baisser les bras et de vous en sortir. La bonne nouvelle est que

- vous savez qu'il y a une (seule) sortie (mais vous ne savez pas où elle est),
- vous savez que le tunnel s'étend sur au plus *limite* mètres à votre gauche, et *limite* mètres à votre droite

A chaque étape, votre seul mouvement possible est "faire un mètre à gauche" (G) ou "faire un mètre à droite " (D). Etant donné qu'il fait noir, vous ne verrez la sortie que si vous êtes exactement devant.

Le but est donc de trouver une séquence de mouvements (par exemple "GDGGDDGGGDDDGGGGDDDD..") qui permette à coup sûr de trouver la sortie, en minimisant la distance totale parcourue.

La situation est modélisée de la façon suivante :

- un entier *limite* ($limite \leq 10000$), saisi par l'utilisateur au début de l'exécution du programme
- un entier *position* (initialisé à 0) qui représente votre position à chaque étape
- un entier *sortie*, tiré au hasard entre $-limite$ et $limite$
- un entier *compteur* (initialisé à 0) qui va être incrémenté à chaque mouvement

Le point clef est que vous allez devoir écrire un algorithme **sans** utiliser la variable *sortie* (puisque vous ne savez pas où elle est), *position* (puisqu'il ne faut pas que vous puissiez faire $position := 18$, ce qui reviendrait à se téléporter) ni *compteur* (puisqu'il ne faut pas que vous trichiez en remettant *compteur* à 0).

Vous avez donc à disposition les fonctions/procédures suivantes

- procedure *init* : qui demande de saisir *limite*, puis tire au sort la position de la sortie dans l'intervalle $[|-limite, ..., limite|]$
- procedure *gauche* : qui vous fait faire un pas à gauche (et incrémente *compteur*)
- procedure *droite* : qui vous fait faire un pas à droite (et incrémente *compteur*)
- fonction *gagne* : qui retourne *true* ssi vous êtes en face de la sortie
- fonction *limite_atteinte* : qui retourne *true* ssi vous êtes à un des bouts du tunnel (il faut donc repartir dans l'autre sens)
- procedure *affiche_etat* : qui affiche le tunnel, votre position et la position de la sortie (vous utiliserez cette procédure en affichant après chaque étape pour vérifier comment se comporte votre algorithme)
- procedure *affcihe_compteur* : qui affiche le compteur !

Dans le tout le TP, vous avez donc seulement le droit d'utiliser ces fonctions/procédures, et évidemment de définir et d'utiliser vos propres variables.

Mise en place

Récupérez le code à l'adresse `http://moais.imag.fr/membres/marin.bougeret/teaching/gd.adb`, et compilez le !

Stratégie du pile ou face

On considère la stratégie suivante : choisir un coté au hasard, et l'explorer jusqu'au bout. Si cela est infructueux, repartir dans l'autre sens jusqu'à trouver la sortie.

Ecrire un algorithme réalisant cette stratégie (pour tirer au hasard, vous utiliserez la fonction $Tirage\_Random(A, B : Integer)$ qui tire à la spécification suivante :

- prérequis : $-10000 \leq A \leq B \leq 10000$
- action : retourne un entier tiré au hasard entre A et B

Stratégie du zig zag

On considère la stratégie suivante : faire un pas à gauche, puis deux à droite, puis trois à gauche, etc .. jusqu'à trouver la sortie.

Ecrire un algorithme réalisant cette stratégie.

A vous !

Est ce que cela à un sens de dire qu'une stratégie est meilleure qu'une autre ?

Testez abondamment : écrivez un algorithme qui répète l'expérience un grand nombre de fois.. (on s'autorisera à lire la valeur de *compteur* d'une expérience à l'autre, afin de pouvoir faire la somme !)

Ecrivez une stratégie à vous, et comparez avec les précédentes en faisant un grand nombre d'expériences.