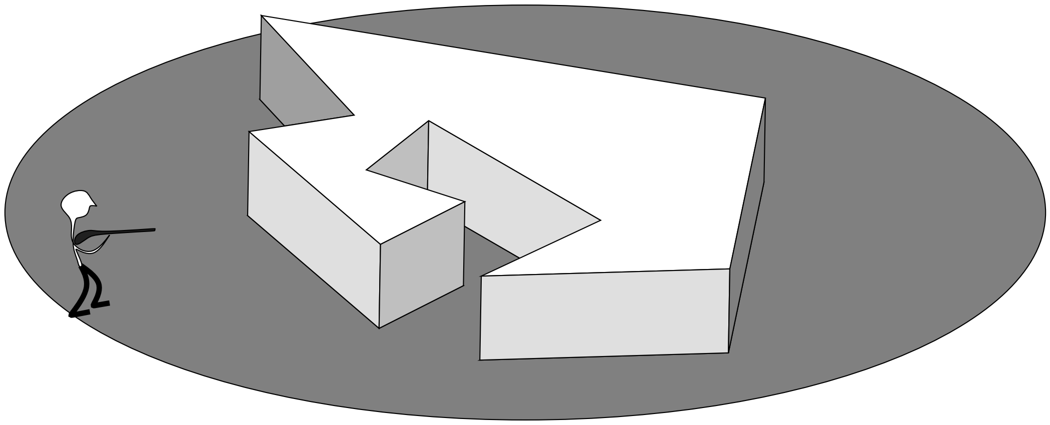
Night paint ball !

Sujet proposé par Olivier Devillers

`Olivier.Devillers[at]sophia.inria.fr`

Difficulté : facile (*) (pour le cas convexe).

1 Préambule : règles du jeu

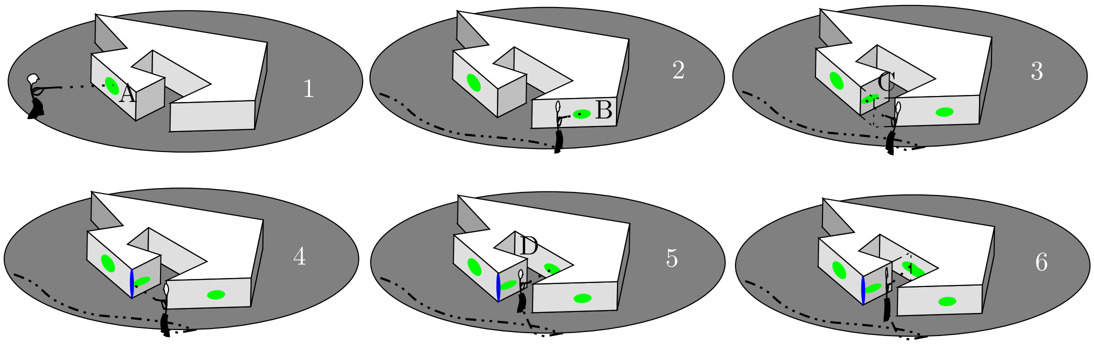


Le tireur se trouve dans le noir face à un «objet» constitué de murs verticaux assemblés. Le tireur est dans le noir, mais il est armé d'un fusil de *paintball* chargé avec de la peinture fluorescente.

Lorsqu'un mur est touché par de la peinture fluorescente, au voisinage de l'impact, il devient visible, c'est à dire que l'on peut déterminer la position et l'orientation de ce mur (sans pour autant connaître ses extrémités). Si on touche exactement un coin, on détermine juste la position du coin (et le fait que c'est un coin).

Le tireur peut se déplacer où il veut à condition de ne pas se cogner. C'est à dire qu'il peut en fait se déplacer sur un grand cercle à l'extérieur dont on sait qu'il contient l'objet, ou bien en se déplacant sur les trajectoires suivies par les balles de peinture dont on sait egalement qu'elles ne rencontre pas l'objet avant le point d'impact.

But du jeu : déterminer tout l'environnement en utilisant le moins de balles possibles.



Dans la figure ci dessus le tireur comme suit :
1— Il tire au hasard vers le centre du cercle et découvre un mur A.
2— De même il découvre un mur B
3— Il essaye de vérifier que A∩B est un coin et découvre à cette occasion un mur C.
4— Il vérifie A∩C
5— Il se déplace sur la trajectoire 3 pour tirer dans la zone inconnue et découvre un mur D
6— Il essaye de vérifier que C∩D est un coin et trouvre à nouveau le mur D.

2 Détail du sujet

C'est un algorithme de simulation, on va avoir deux parties du programme qui se repondent, celle qui ne connait pas l'objet et essaye de le découvrir en tirant dessus, et celle qui connait l'objet et qui doit donner le résultat des tirs. Ces deux parties doivent être clairement séparées et communiquer d'une manière bien définie.

2.1 Interface

On prévoiera un interface graphique 2D «vue de dessus». Initiallement on affichera le cercle et le centre du cercle, l'utilisateur pourra alors saisir à la souris un polygone inclu dans le cercle contenant le centre du cercle. On affichera ensuite la recherche du polygone en affichant les trajectoires des balles, celle du tireur, et les parties connues du polygone dans des couleurs différentes. On pourra prévoir un affichage pas à pas (on clique pour chaque nouvelle balle) et quelque chose de plus automatique.

2.2 Calcul des impacts et représentation du polygone

Le polygone pourra être représenté par une liste de points. Le calcul de l'impact d'une balle sur le polygone sera alors calculé en calculant l'intersection de la demi-droite parcourue par la balle avec chaque coté du polygone et en sélectionnant la plus proche de l'origine de la demi-droite. Attention, pour une balle tirée sur un coin, les erreurs d'arrondi peuvent poser problèmes (par exemple la balle passe entre les deux murs théoriquement jointifs !), vous êtes libre de proposer toute solution pour régler ces problèmes.

2.3 Algorithme pour un objet convexe

1- On tire sur le centre du cercle depuis 2 points opposés sur le cercle. Ce qui nous permet de déterminer deux droites supports d'arêtes de l'objet. Ces deux droites se coupent en I, on tire alors en utilisant la droite passant par I et le centre du cercle de manière à rencontrer le centre du cercle avant I. On détermine ainsi une troisième droite support, et on a une première approximation de l'objet en considérant le triangle formé par ces trois droites.

2- Tant qu'il y a des arêtes de l'objet dont on ne connait pas les extrémités, on tire sur l'intersection de deux arêtes consécutives, ce qui nous permet soit de vérifier un coin, soit de découvrir une nouvelle arête que l'on insère entre les deux précédentes.

Vous devrez maintenir la liste des arêtes partiellement connues dans l'ordre du bord de l'objet. À chaque tir, soit on découvre une nouvelle arête, soit on vérifie un coin, la complexité de cet algorithme est donc clairement de $2n$ tirs.

3 Pour aller plus loin

3.1 Calcul des impacts plus rapide

L'essentiel de l'algorithme qui simule la réponse aux tirs consiste à calculer l'intersection d'un rayon avec un polygone convexe. Plutôt que de tester toutes les arêtes, il est plus judicieux de procéder par dichotomie dans le cas convexe. On peut alors représenter le polygone par un arbre binaire dont chaque feuille correspond à une arête et trouver la bonne arête en cherchant dans cet arbre. Attention, comme le polygone est intrinsèquement une liste circulaire, on risque en fait dans certain cas de faire deux recherches dans l'arbre.

3.2 Algorithme pour un objet non convexe

L'initialisation est identique à celle du cas convexe.

Pour la suite, soit A et B les droites supports de deux arêtes consécutives. Soit $I = A \cap B$. On tire sur I et on a alors trois cas :
–1– On confirme que I est un coin (on a trouvé une extrémité de A et de B)
–2– On trouve une autre arête C que l'on insère entre A et B
–3– On trouve A (ou B, c'est ce qui est nouveau par rapport au cas convexe, regardez la figure à l'étape 6). Dans ce cas on est sur que l'arête B est stoppée avant I, et donc si on tire sur I depuis l'autre coté, on va trouver une autre arête. Attention il faut chaoisir les positions de tirs dans la zone connue.

Vous pouvez trouver plus de détails dans cet article [ABY88].

Références

[ABY88] P. Alevizos, J.-D. Boissonnat, and M. Yvinec. On the order induced by a set of rays. application to the probing of non convex polygons. Technical Report 927, INRIA, 1988. `http://www.inria.fr/rrrt/rr-0927.html` `http://www.inria.fr/rrrt/rr-0927.html`.