

# CHALLENGE INFORMATIQUE - BATAILLE DE BOULE DE NEIGE

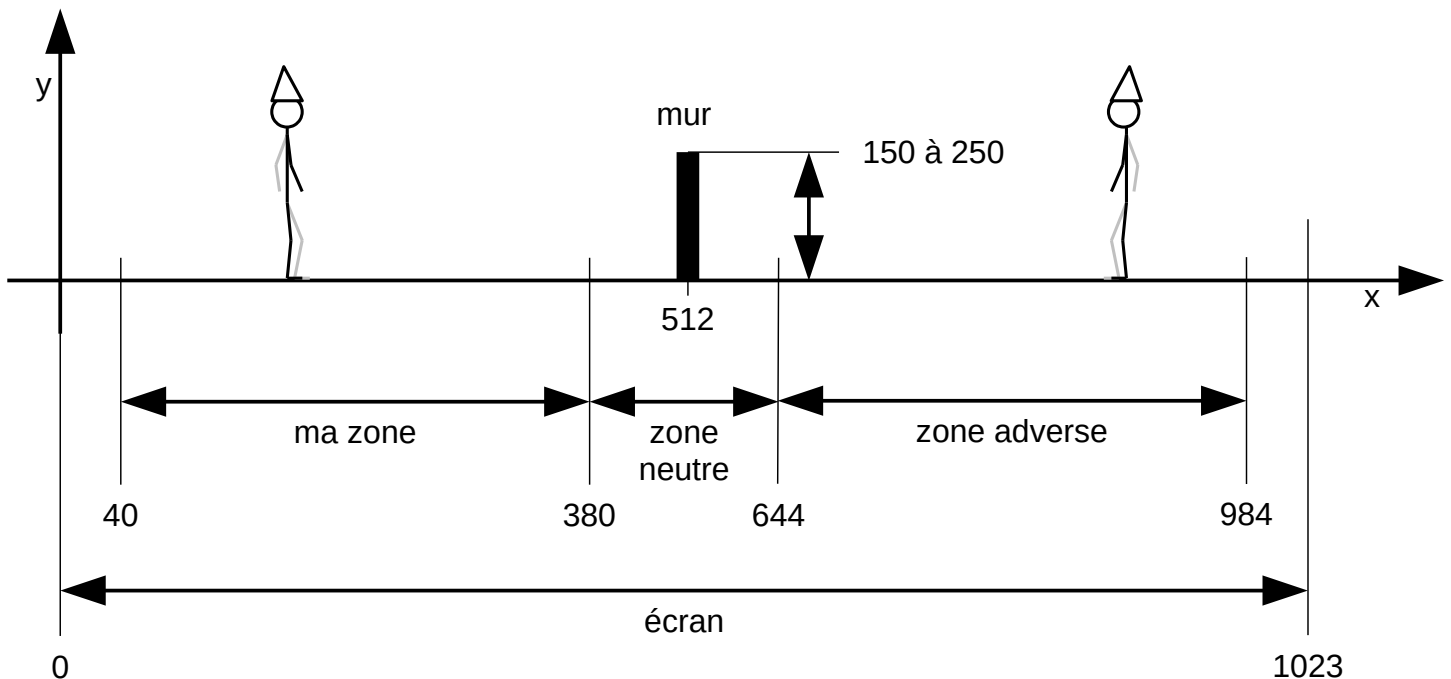
## Règlement 2020

### PRINCIPE GÉNÉRAL

Le match se déroule dans un espace à 2 dimensions.

Il y a deux participants (robots humanoïdes virtuels). Le jeu consiste à envoyer des boules de neige sur le robot adverse. Chaque robot a un camp. Il existe au milieu du terrain une zone neutre où aucun des deux robots ne peut se trouver. Cette zone est infranchissable. Au milieu de cette zone neutre se trouve un mur/bonhomme de neige de hauteur variable d'un match à l'autre (150 à 250 pixels). Un pixel correspond à environ 5mm.

L'aire de jeu avec une échelle en pixels :



Au début de la partie, le sol est recouvert d'une hauteur aléatoire de neige. La neige peut tomber au cours de la partie.

### TRAVAIL A FAIRE

Votre programme doit se connecter à un serveur et donner des ordres au robot (se déplacer, ramasser de la neige, lancer une boule...). Le serveur assure l'affichage de la partie et la vérification des informations envoyées.

Des fichiers d'exemple sont fournis. Cela vous permettra de ne pas avoir à programmer la connexion au serveur, le téléchargement des images de votre robot et le démarrage du jeu. L'exemple propose également une gestion basique du robot pour vous aider.

Vous avez à programmer l'automatisation de votre robot, c'est-à-dire son intelligence.

## UN ROBOT

Un robot debout a une hauteur de 225 pixels avec son bonnet. Il se décompose en 4 parties :

- le bonnet
- la tête
- le tronc
- les jambes

## DÉROULEMENT D'UNE PARTIE

Chaque partie aura un temps déterminé (environ 1 minute). Au début de la partie, le robot, avec son bonnet sur la tête, dispose de points de vie (entre 70 et 160 points, cf. paragraphe POINTS DE QUALITÉ) qu'il perd quand il reçoit une boule de neige. Lorsqu'un robot n'a plus de points de vie avant la fin du temps de jeu, il est battu (mis KO), et la partie s'arrête.

Boule reçue dans	Points de vie perdus
le bonnet	0
la tête	10
le tronc	2
les jambes	1

En plus de ses points de vie, chaque robot marque des points de score s'il touche son adversaire. Chaque fois qu'une boule de neige est lancée, le score augmente d'un point. Si une boule atteint l'adversaire, le score augmente suivant le point d'impact. A noter que si la boule de neige touche le bonnet et que l'impact est suffisamment fort, alors le bonnet tombe et reste au sol pour le restant de la partie (il n'est pas possible de ramasser le bonnet).

	Score obtenu
boule lancée	1
boule reçue dans le bonnet (s'il tombe)	100
boule reçue dans la tête	20
boule reçue dans le tronc	5
boule reçue dans les jambes	1

Si l'adversaire est mis KO, c'est-à-dire qu'il n'a plus de point de vie avant la fin du temps réglementaire, le robot reçoit un bonus de 100 points plus 5 points par seconde restante dans la partie.

Si la fin de partie est atteinte, au bout d'un temps déterminé, sans qu'aucun robot ne soit KO, c'est le score obtenu qui décide du vainqueur.

Dans le cas particulier d'un impact à la tête, le robot peut être sonné et le reste un certain temps dépendant de l'énergie de la boule de neige reçue. Plus la boule de neige aura de l'énergie (énergie cinétique) plus le robot restera sonné longtemps.

## ACTIONS

Le robot peut effectuer plusieurs actions. Chaque action dure un temps standard.

Action	Durée
Avancer (pour 20 pixels)	0,1 s
Reculer (pour 20 pixels)	0,1 s
S'accroupir	0,4 s
Rassembler de la neige ( <b>seulement si le robot est accroupi</b> )	0,5 s
Compacter de la neige pour fabriquer une boule de neige, la durée dépend de la quantité de neige utilisée pour faire la boule ( <b>seulement si la neige a été rassemblée et si le robot est accroupi</b> )	0,3 s à 0,75 s
Se relever	0,5 s
Lancer une boule de neige	0,7 s
Stopper son action (par exemple arrêter d'avancer)	x

## POINTS DE QUALITÉ

Le robot dispose de 10 points de qualité à répartir dans 4 caractéristiques : l'**endurance**, la **rapidité**, la **force** et la **précision**. Chaque caractéristique peut varier de 0 à 5 points.

L'**endurance** permet d'augmenter le nombre de points de vie.

Endurance	Points de vie
0	70
1	90
2	100
3	110
4	130
5	160

La **rapidité** permet de modifier le temps mis pour effectuer chaque action.

Rapidité	Modification du temps standard des actions
0	+33 %
1	+11 %
2	0 %
3	-9 %
4	-20 %
5	-33 %

La **force** permet de lancer les boules avec plus d'énergie. Attention, ceci a surtout un impact sur les boules de neige lourdes car la vitesse initiale maximale du lancer est limitée à 1500 pixels/s.

Force	Modification de l'énergie maximale du lancé	Énergie disponible par rapport à l'énergie standard fixée à 100 %
0	-25 %	0 à 75 %
1	-10 %	0 à 90 %
2	0 %	0 à 100 %
3	+10 %	0 à 110 %
4	+25 %	0 à 125 %
5	+50 %	0 à 150 %

La **précision** permet d'effectuer des lancers de boule de neige plus précis.

Précision	Erreur d'angle	Erreur de force
0	$\pm 4^\circ$	$\pm 10 \%$
1	$\pm 2^\circ$	$\pm 7,5 \%$
2	$\pm 1^\circ$	$\pm 5 \%$
3	$\pm 0,5^\circ$	$\pm 2,5 \%$
4	$\pm 0,25^\circ$	$\pm 1 \%$
5	$\pm 0^\circ$	$\pm 0 \%$

## NEIGE

La neige est présente sur le sol. Elle peut tomber au cours de la partie. Le robot connaît la quantité de neige disponible au sol à l'endroit où il se trouve s'il se baisse pour la rassembler.

## UNE BOULE DE NEIGE

Toutes les boules de neige ont la même taille (rayon d'environ 5cm soit 10 pixels).

Pour fabriquer sa boule, le robot peut compacter plus ou moins de neige. La neige nécessaire pour fabriquer une boule varie de 20 points de neige pour une boule peu compacte à 50 points de neige pour une boule très compacte. La masse d'une boule de neige varie linéairement de 100g (20 points de neige) à 250g (50 points de neige) en fonction de la quantité de neige compactée.

Le temps de fabrication est proportionnel à la quantité de neige à compacter (0,15 s pour 10 points de neige).

Plus une boule est compacte, plus elle nécessite d'énergie pour être lancée vite (et donc arriver loin). Autrement dit une même énergie de départ une boule lourde ira moins loin qu'une boule légère.

Le robot ne peut pas avoir en main plus d'une seule boule de neige.

A chaque instant, le robot connaît les positions (en pixels) et les vitesses (en pixels/s) de toutes les boules de neige qui sont en l'air.

## **LE LANCÉ D'UNE BOULE DE NEIGE**

On considère que l'énergie est transmise intégralement à la boule de neige à l'erreur de précision près. On détermine la vitesse initiale de la boule de neige en considérant son énergie cinétique initiale. On majore alors la vitesse initiale à 1500 pixels/s.

## **VENT**

Le vent modifie les trajectoires des boules de neige. Il varie au cours de la partie. Il peut souffler dans toutes les directions. Sa composante suivant x peut varier de -1500 à 1500 pixels/s. Sa composante suivant y peut varier de -300 à 300 pixels/s.

Le frottement d'une boule de neige dans l'air est assimilé à une force proportionnelle au carré de la vitesse relative de la boule, inversement proportionnelle à la densité de la boule et opposée au vecteur vitesse relative de la boule.

## **LES INFORMATIONS DE MON ROBOT**

Les informations suivantes sont disponibles concernant mon robot :

- ses points de vie restants
- son score actuel
- sa position
- s'il a encore son bonnet sur la tête
- s'il a une boule de neige en main
- la neige disponible pour lui s'il se baisse
- si la neige a été rassemblée
- ce qu'il est en train de faire
- s'il est sonné et si oui pour combien de temps

## **L'ADVERSAIRE**

Les informations suivantes sont disponibles concernant l'adversaire :

- ses quatre caractéristiques (endurance, rapidité, force, précision)
- ses points de vie restants
- son score actuel
- sa position
- s'il a encore son bonnet sur la tête
- s'il a une boule de neige en main
- la neige disponible pour lui s'il se baisse
- ce qu'il est en train de faire
- s'il est sonné et si oui pour combien de temps

## **PARAMÈTRES GÉNÉRAUX DU JEU**

Le temps est découpé en intervalles de 20 ms.

Les informations sont disponibles à chaque pas de temps :

- Le temps restant avant la fin de la partie
- La hauteur du mur
- La vitesse du vent suivant les axes x et y
- la présence d'une chute de neige

## LE DESIGN DU ROBOT

La représentation du robot doit être modifiée.

Votre rôle est de **modifier les images** qui permettent les animations du robot. Vous devez utiliser obligatoirement la transparence et le format **png**.

**ATTENTION**, la taille des images initiales doit impérativement être conservée.

Au total, un ensemble de 8 images permettent la représentation complète du robot.

## LE SERVEUR

L'application serveur veille à l'équité du match.

Quand l'application serveur est démarrée, deux clients peuvent se connecter pour jouer un match.

Votre rôle est d'**écrire le code de votre client pour commander les actions du robot** et remporter la victoire.

Touche	Fonction
Échap ou Q	Quitter l'application
F11 ou F	Passer du mode « Plein écran » au mode fenêtré et inversement
F10	Grande fenêtre
F9	Moyenne fenêtre
F8	Petite fenêtre
R	Remettre à zéro
S	Démarrer un match
+	Augmenter la durée du match de 10 s
-	Diminuer la durée du match de 10 s
Flèche de gauche	Régler l'IA du robot de gauche
Flèche de droite	Régler l'IA du robot de droite
Flèche du haut	Augmenter le niveau de l'IA
Flèche du bas	Diminuer le niveau de l'IA

## LE CLIENT

Il est capable de transférer les images sur le serveur, de donner un nom au robot, de changer les caractéristiques du robot et de contrôler les actions du robot. Une bibliothèque de fonctions compatibles avec l'application serveur est fournie. Les fonctions de cette bibliothèque sont documentées ci-après.

## FONCTIONS DE CONNEXION ET DE DÉCONNEXION

**int serveurConnecter(char \*adresse, unsigned short int port);**

Rôle de la fonction :

Effectuer une connexion à un serveur.

Paramètres :

char \*adresse

Adresse IP du serveur sous forme d'une chaîne de caractère. Ex : "192.168.0.1"

En local vous pouvez utiliser l'adresse "127.0.0.1"

int port

Numéro du port sur lequel le serveur écoute. Ex : 1050 valeur par défaut du serveur.

Valeur de retour :

SERVEUR\_ERREUR\_ID

Erreur d'identification.

SERVEUR\_INTROUVABLE

Erreur serveur introuvable.

SERVEUR\_CONNECTE

Connexion au serveur réussie.

**int serveurFermer(void);**

Rôle de la fonction :

Fermer la connexion au serveur en cours.

Valeur de retour :

1 Déconnexion réussie.

## FONCTION DE DÉMARRAGE D'UN MATCH

**int serveurDemarrerMatch(void);**

Rôle de la fonction :

Démarrer un match créé sur le serveur. Fonction disponible en mode test. Non disponible en mode compétition car c'est alors le serveur qui décide du début du match.

Valeur de retour :

1 Envoi réussi.

0 Échec de l'envoi.

## FONCTIONS DE MODIFICATION DU ROBOT AVANT UN MATCH

**int serveurUpload(int num, char \*nom);**

Rôle de la fonction :

Envoyer un fichier image au format **png** vers le serveur pour modifier la représentation du robot.

Paramètres :

int num

Numéro de l'image concernée. Les constantes suivantes, disponibles dans la bibliothèque, devraient être utilisées :

#define IMAGE_DEBOUT	0
#define IMAGE_MARCHE	1
#define IMAGE_ACCROUPI	2
#define IMAGE_LANCE	3
#define IMAGE_BONNET_PROFIL	4
#define IMAGE_VICTOIRE	5
#define IMAGE_DEFAITE	6
#define IMAGE_BONNET_FACE	7

char \*nom

Nom du fichier image entre " ". Ex : "../images/debout.png"

Valeur de retour :

1 Envoi réussi.

0 Échec de l'envoi.

```
int serveurNomRobot(char *nom) ;
```

Rôle de la fonction :

Nommer le robot sur l'application serveur.

Paramètres :

```
char *nom
```

Nom du robot. Au maximum 15 caractères sont pris en compte.

Ex : "Mon robot"

Valeur de retour :

1 Envoi réussi.

0 Échec de l'envoi.

```
int serveurCaracRobot(int rapidite, int precision, int force, int endurance) ;
```

Rôle de la fonction :

Modifier les caractéristiques du robot. Chaque caractéristique est un entier entre 0 et 5. La somme des caractéristiques ne doit pas dépasser 10 pour que la modification soit prise en compte.

Paramètres :

```
int rapidite 0 à 5
```

```
int precision 0 à 5
```

```
int force 0 à 5
```

```
int endurance 0 à 5
```

Valeur de retour :

1 Envoi réussi.

0 Échec de l'envoi.

## FONCTIONS DE CONTROLE DES ACTIONS

```
int serveurAvancer(void) ;
```

Rôle de la fonction :

Fais avancer le robot. Le robot doit être dans un état immobile pour que l'ordre soit pris en compte.

Sans un ordre d'arrêt, le robot continue à avancer jusqu'à la limite de sa zone.

Valeur de retour :

1 Envoi réussi.

0 Échec de l'envoi.

```
int serveurReculer(void) ;
```

Rôle de la fonction :

Fais reculer le robot. Le robot doit être dans un état immobile pour que l'ordre soit pris en compte.

Sans un ordre d'arrêt, le robot continue à avancer jusqu'au début de sa zone.

Valeur de retour :

1 Envoi réussi.

0 Échec de l'envoi.

```
int serveurSAccroupir(void) ;
```

Rôle de la fonction :

Fais s'accroupir le robot. Le robot doit être dans un état immobile ou en train de se relever pour que l'ordre soit pris en compte. Cet ordre ne peut être interrompu que par un ordre de se relever.

Valeur de retour :

1 Envoi réussi.

0 Échec de l'envoi.

```
int serveurSeRelever(void) ;
```

Rôle de la fonction :

Fais se relever le robot. Le robot doit être dans un état accroupi ou en train de s'accroupir pour que l'ordre soit pris en compte. Cet ordre ne peut être interrompu que par un ordre de s'accroupir.

Valeur de retour :

1 Envoi réussi.

0 Échec de l'envoi.



**int serveurRassemblerNeige(void) ;**

Rôle de la fonction :

Donne l'ordre de rassembler de la neige. Le robot doit être dans un état accroupi pour que l'ordre soit pris en compte. Cet ordre ne peut être interrompu que par un ordre de stopper l'action. Quand la neige est rassemblée, le robot revient automatiquement en position accroupie. Si cet ordre est interrompu avant sa fin, le travail effectué est perdu.

Valeur de retour :

1      Envoi réussi.  
0      Échec de l'envoi.

**int serveurNeRienChanger(void) ;**

Rôle de la fonction :

Ne fais rien et poursuit l'action en cours.

Valeur de retour :

1      Envoi réussi.  
0      Échec de l'envoi.

**int serveurStopperAction(void) ;**

Rôle de la fonction :

Interrompt l'action en cours si c'est une action qui peut être stoppée.

Valeur de retour :

1      Envoi réussi.  
0      Échec de l'envoi.

**int serveurCompacterNeige(int neige) ;**

Rôle de la fonction :

Donne l'ordre de compacter la neige. Si la neige n'a pas été rassemblée ou si la quantité de neige est insuffisante, l'action n'est pas effectuée. De plus, le robot doit être dans un état accroupi pour que l'ordre soit pris en compte. Cet ordre ne peut être interrompu que par un ordre de stopper l'action. Quand la neige est compactée, le robot possède une boule de neige et il revient automatiquement en position accroupie. Si cet ordre est interrompu avant sa fin, le travail effectué est perdu. La quantité de neige reste au sol.

Paramètres :

int neige                      quantité de neige à compacter de **20 à 50**

Valeur de retour :

1      Envoi réussi.  
0      Échec de l'envoi.

**int serveurLancer(double energie, double angle) ;**

Rôle de la fonction :

Lance une boule de neige. Le robot doit être en position immobile pour que l'action soit exécutée. Si le robot ne possède pas de boule de neige, l'action est exécutée mais aucune boule n'est lancée. Cette action ne peut pas être interrompue.

Paramètres :

double energie

Énergie mise dans le lancé en % de **0.0 à 150.0** % si le robot a une force de 5

double angle

Angle de lancé en degré de **-90.0° à +90.0°**

Valeur de retour :

1      Envoi réussi.  
0      Échec de l'envoi.

## FONCTION DE RÉCUPÉRATION DES DONNÉES TRANSMISES PAR LE SERVEUR

```
int serveurRecevoirSituation(Jeu *pJeu, Moi *pMoi, Adversaire *pAdversaire,  
                             int *pNbBoules, Boule *pBoules);
```

Rôle de la fonction :

Récupère l'état actuel du jeu. Pendant un match, cette fonction devrait être exécutée en boucle.

Paramètres :

Jeu \*pJeu

Pointeur vers une structure pour récupérer l'état du jeu. Les informations contenues sont détaillées ci-après.

Moi \*pMoi

Pointeur vers une structure pour récupérer l'état de mon robot. Les informations contenues sont détaillées ci-après.

Adversaire \*pAdversaire

Pointeur vers une structure pour récupérer l'état du robot adverse. Les informations contenues sont détaillées ci-après.

int pNbBoules

Pointeur vers un entier pour récupérer le nombre de boules en vol.

Boule \*pBoules

Pointeur vers un tableau pour récupérer les informations de toutes les boules en vol.

Valeur de retour :

1 Réception réussi.

0 Échec de la réception.

## VARIABLE DE TYPE JEU

Jeu jeu;

jeu.etat

JEU\_ATTENTE

Les fonctions de modification du robot peuvent être utilisées

JEU\_MATCH\_EN\_COURS

Les fonctions de contrôles des actions peuvent être utilisées

JEU\_FIN\_MATCH

La partie est terminée

jeu.nbJoueursCnx

1 ou 2, permet de savoir si le match a lieu contre un autre client ou contre le serveur

jeu.chrono

Temps restant dans la partie en millisecondes

jeu.ventX

Composante x du vent exprimée en pixels/s entre -1500 pixels/s et +1500 pixels/s. Un vent négatif souffle face au robot.

jeu.ventY

Composante y du vent exprimée en pixels/s entre -300 pixels/s et +300 pixels/s. Un vent négatif souffle vers le sol.

jeu.hauteurMur

Hauteur du mur en pixels.

jeu.ilNeige

1 si la neige est en train de tomber et 0 sinon.

## VARIABLE DE TYPE MOI

**Moi moi;**

**moi.etat**

Valeurs possibles pendant un match :

ROBOT\_IMMOBILE  
ROBOT\_AVANCE  
ROBOT\_RECULE  
ROBOT\_S\_ACCROUPI  
ROBOT\_ACCROUPI  
ROBOT\_RASSEMBLE\_NEIGE  
ROBOT\_COMPACTE\_BOULE  
ROBOT\_SE\_RELEVE  
ROBOT\_LANCE

Valeurs possibles à la fin d'un match :

ROBOT\_VICTOIRE  
ROBOT\_DEFAITE

**moi.x**

Position x. Le robot occupe toujours le camp de gauche. x peut varier de ROBOT\_XMIN (40) à ROBOT\_XMAX (380).

**moi.bonnet**

0 si le bonnet est tombé, 1 si le bonnet est sur la tête

**moi.nbBoule**

1 si le robot possède une boule, 0 sinon

**moi.neigeDispo**

Quantité de neige disponible immédiatement devant le robot

**moi.neigeRassemblee**

1 si la neige a été rassemblée, 0 sinon

**moi.ptsDeVie**

Nombre de points de vie restants

**moi.score**

Score actuel

**moi.blackOut**

0 si le robot n'est pas sonné, sinon nombre de millisecondes avant que le robot recommence à fonctionner

**moi.reponse**

1 si la dernière action demandée a été prise en compte, 0 sinon

## VARIABLE DE TYPE ADVERSAIRE

**Adversaire** **adversaire;**

**adversaire.etat**

Valeurs possibles pendant un match :

ROBOT\_IMMOBILE  
ROBOT\_AVANCE  
ROBOT\_RECULE  
ROBOT\_S\_ACCROUPI  
ROBOT\_ACCROUPI  
ROBOT\_RASSEMBLE\_NEIGE  
ROBOT\_COMPACTE\_BOULE  
ROBOT\_SE\_RELEVE  
ROBOT\_LANCE

Valeurs possibles à la fin d'un match :

ROBOT\_VICTOIRE  
ROBOT\_DEFAITE

**adversaire.x**

Position x. Le robot adverse occupe toujours le camp de droite.

**adversaire.bonnet**

0 si le bonnet est tombé, 1 si le bonnet est sur la tête

**adversaire.nbBoule**

1 si le robot possède une boule, 0 sinon

**adversaire.neigeDispo**

Quantité de neige disponible immédiatement devant le robot

**adversaire.ptsDeVie**

Nombre de points de vie restants à l'adversaire

**adversaire.score**

Score actuel de l'adversaire

**adversaire.blackOut**

0 si le robot adverse n'est pas sonné, sinon nombre de millisecondes avant que le robot adverse recommence à fonctionner

**adversaire.rapidite**

Caractéristique de rapidité du robot adverse entre 0 et 5

**adversaire.precision**

Caractéristique de précision du robot adverse entre 0 et 5

**adversaire.force**

Caractéristique de force du robot adverse entre 0 et 5

**adversaire.endurance**

Caractéristique d'endurance du robot adverse entre 0 et 5

## VARIABLE DE TYPE BOULE

**Boule boule;**

**boule.x**

Coordonnée x du centre de la boule en pixels. L'axe des x va de gauche à droite.

**boule.y**

Coordonnée y du centre de la boule en pixels. L'axe des y va du bas vers le haut. Le sol est à 0.

**boule.vx**

Composante x de la vitesse de la boule en pixels/s

**boule.vy**

Composante y de la vitesse de la boule en pixels/s

Exemple d'interrogation du serveur :

**Jeu jeu;**

**Moi moi;**

**Adversaire adversaire;**

**int nbBoules;**

**Boule boule[BOULES\_NB\_MAX];**

**serveurRecevoirSituation(&jeu, &moi, &adversaire, &nbBoules, boule);**

Si `nbBoules` vaut 2 alors il y a deux boules en vol. La première a pour coordonnées

`boule[0].x`

`boule[0].y`

et la seconde a pour coordonnées

`boule[1].x`

`boule[1].y`