CyberCycles

Défi

Cette année, vous devrez programmer des intelligences artificielles qui sauront combattre leurs adversaires dans un environnement inspiré de TRON.

À quatre personnes, vous devrez programmer *deux intelligences artificielles* qui se battront en équipe contre deux autres intelligences artificielles dans un tournoi à mort.

Vous pouvez donc choisir d'utiliser une approche classique, où chaque robot devra essayer de survivre jusqu'à la fin de la partie, ou vous pouvez tenter de faire collaborer vos deux robots !

Directives

La première chose à effectuer est de choisir une valeur pour `ROOM`, commune aux membres de votre équipe.

Par la suite, pour pouvoir tester votre AI, vous devez démarrer quatre instances, dans la même chambre. Chaque 2 instances doivent avoir la même équipe (`TEAM`).

Vos modifications doivent principalement se faire dans `AI.java`, mais vous pouvez vous créer d'autres fichiers au besoin.

Structure

Début de partie

Au début de la partie, `start` est appelée. Vous recevez un objet en *JSON* qui contient la configuration du jeu, générée par le serveur.

Voici un exemple de ce `JSONObject` :

```
{
  "players": [{
    "x": 1,
    "y": 19,
    "id": "1",
    "team": "1",
    "direction": "u"
  }, {
    "x": 4,
    "y": 17,
    "id": "2",
```

```
    "team": "1",
    "direction": "d"
  }, {
    "x": 30,
    "y": 21,
    "id": "3",
    "team": "2",
    "direction": "u"
  }, {
    "x": 18,
    "y": 21,
    "id": "4",
    "team": "2",
    "direction": "d"
  }],
  "obstacles": [{
    "w": 3,
    "x": 37,
    "h": 4,
    "y": 13
  }, {
    "w": 3,
    "x": -2,
    "h": 4,
    "y": 12
  }],
  "w": 38,
  "h": 29,
  "me": "1"
}
```


`players` est un tableau qui contient des objets représentant des joueurs, avec leurs coordonnées de départ (`x` et `y`), leur identifiant (`id`), leur équipe (`team`) et leur direction (`direction`).

`obstacles` est un tableau qui contient des objets représentant des obstacles rectangulaires sur la grille de jeu, avec leurs coordonnées (`x` et `y`), leurs dimensions (`w` sur l'axe des x et `h` sur l'axe des y).

Finalement, `w` représente la longueur de la grille, `h` représente la largeur de la grille et `me` représente votre propre identifiant.

À chaque tour de jeu

À chaque tour de jeu, `next` est appelée. Vous recevez alors un tableau en *JSON* qui contient les mouvements précédents des joueurs, tels que reçus par le serveur.

Le `JSONArray` ressemble à ce qui suit :

```
[{
  "id": "1",
  "direction": "u"
}, {
  "id": "2",
  "direction": "u"
}]
```

Chaque `JSONObject` contient l'identifiant du joueur (`id`) et sa direction prise durant le dernier tour de jeu (`direction`).

`direction` doit être retournée dans `next` suite à la décision prise par votre intelligence artificielle, à chaque tour de jeu.

`direction` est une `String` qui n'accepte que les valeurs suivantes :

- `u` pour **up** (haut)
- `l` pour **left** (gauche)
- `d` pour **down** (bas)
- `r` pour **right** (droite)

Fin de partie

À la fin de la partie, `end` est appelée avec, comme attribut, l'identifiant de l'équipe qui a gagné.

Il n'est pas nécessaire de faire quelque chose avec la valeur, mais ça peut vous être utile durant les tests.

Détails techniques

Temps d'exécution

Vous ne disposez pas d'un temps infini pour calculer la prochaine direction dans laquelle vous souhaitez vous déplacer.

Si votre robot ne termine pas l'exécution de la fonction `next` avant un certain délai déterminé par le serveur, on assumera que vous souhaitez continuer en ligne droite.

Votre robot s'imaginera aller dans la direction voulue (demandée en retard), ce qui risquera de causer votre défaite dans les tours qui suivent.

`JSONObject`

Comme mentionné plus haut, quelques réponses du serveur, sont envoyées en *JSON*.

La classe `JsonObject` permet de gérer tout ça, avec des méthodes telles que :

- `getInt`
- `getBoolean`
- `getString`
- ...

Vous pouvez consulter la documentation complète sur internet.