Compétition d’intelligence artificielle

Contexte

Chaque année, dans la ville PetitCarréDeSable, une guerre fait rage. Les meilleurs informaticiens du monde se réunissent, armés de leurs chars d'assaut, pour démontrer leur savoir faire en programmation d'intelligence artificielle. Le but est de combattre les chars d'assaut de l'équipe adverse. L'équipe qui restera à la fin sera déclarée vainqueur.

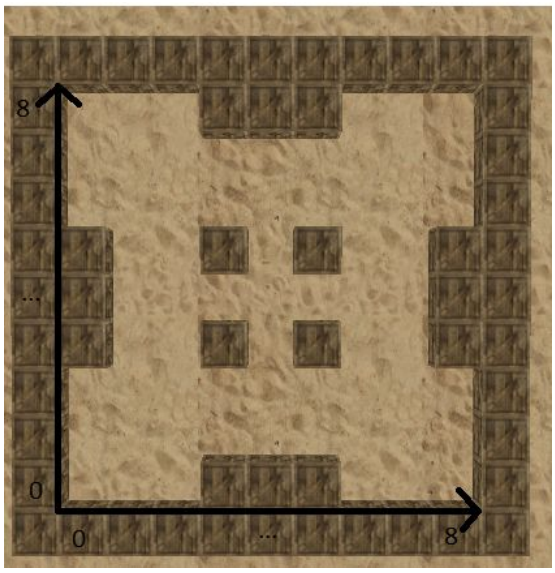
Description du jeu

Deux équipes contrôlant chacun **deux char d'assaut** vont essayer **d'élimer l'équipe adverse**. Vous devez alors implémenter l'intelligence artificielle des deux chars d'assaut que contrôle votre équipe pour élimer ceux de l'équipe adverse. Votre équipe sera alors confrontée aux autres équipes qui participent à la compétition.

Il est à noter qu'une intelligence artificielle non-fonctionnelle (boucle infinie, code qui plante) donnera une victoire par forfait à l'équipe adverse. Comme le jeux est en temps réelle, votre I.A. peut prendre autant de temps qu'elle veut pour calculer ses ordres. Cependant l'I.A. la plus rapide a l'avantage de donner ses ordres avant l'autre. Tous les calculs sont fait côté serveur.

Description du monde

La taille de la carte est fixe et est clôturée par des boites qui empêche les chars d'assaut de sortir. La carte fait 9 cases de long par 9 cases de haut pour un total de 81 case possible. Un total de 16 cases sont bloquées par des boites qui ne peuvent être déplacées ce qui diminue le nombre de cases libres à 65. Les coordonnées des cases sont données par les axes indiquées sur la photo suivante.

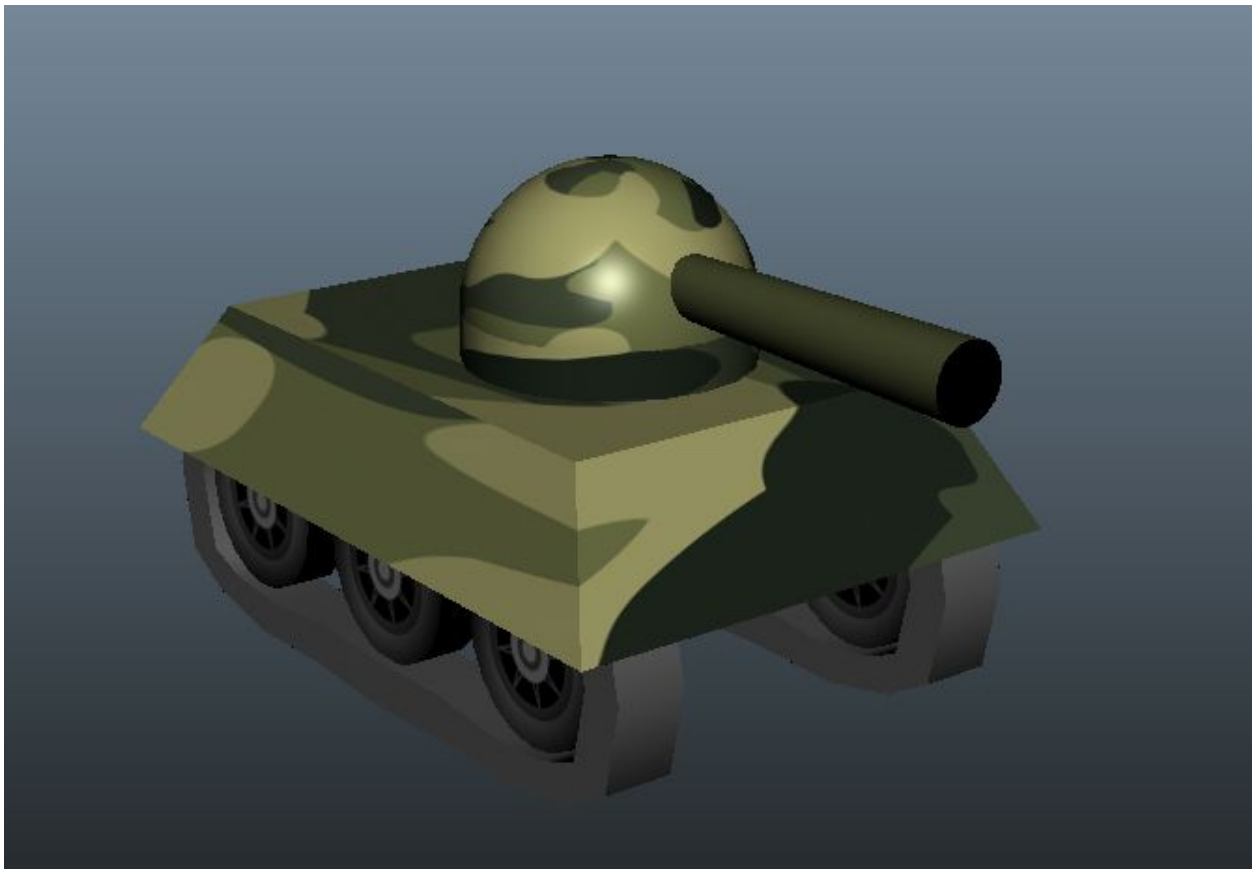


Équipe

Chaque équipe contrôle 2 chars d'assaut. Tant que l'un des chars d'assaut reste en vie, l'équipe peut continuer d'envoyer des commande à celui-ci.

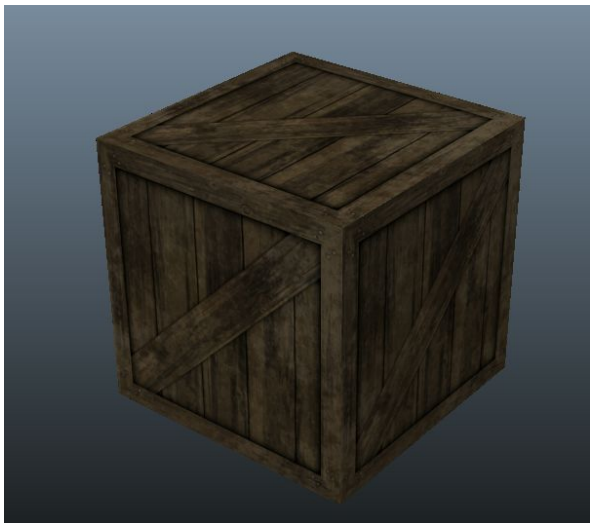
Char d'assaut

Les chars d'assaut exécutent les ordres individuels donnés. La position de départ de chacun des char d'assaut est aléatoirement choisie parmi les coins de la carte, c'est à dire les points: [(0,0), (0,7), (7,0), (7,7)]. Chaque char commence avec un total de **3 points de vie**. Le nombre de point de vie ne peut pas être augmenté durant la partie. Lorsque le char entre en collision avec une mine ou un missile, celui-ci perds un point de vie. Les actions que le char peut exécuter seront décrites dans la section "Action possible pour l'I.A". **Un seul char d'assaut peut occuper une case**. Si un char avait pour ordre de bouger vers une case déjà occupée, celui-ci va essayer de trouver un chemin alternatif. Il n'y a donc **aucune collision entre char** possible.



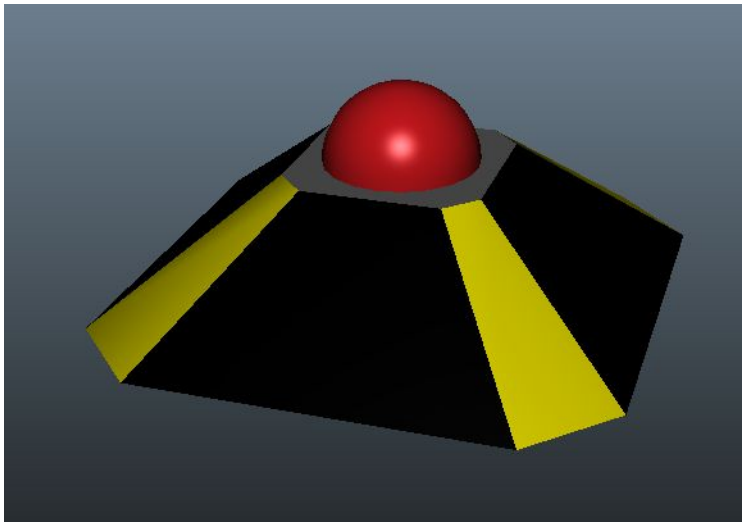
Boite

Les boites ne bougent pas durant la partie et leurs positions sont connues par les deux équipes. L'utilisation de la fonction "isBoxAtPosition" présente dans l'objet "World" vous permet de savoir si une boite se trouve sur une certaine case.



Mines

Les mines sont déposées par les chars et **n'apparaissent pas sur la carte** une fois déposées. Il faut donc faire attention à où on met les pieds. **Une seule** mine peut être déposée par char d'assaut. Un char d'assaut pourra mettre une mine seulement lorsque la dernière mine qu'il a déposée est détruite. Les mines peuvent seulement être déposée sur les cases de la carte. Si l'on donne l'ordre à un char de déposer une mine et que celui-ci est entre deux cases, la mine sera déposée seulement lorsque le char sera complètement sur la case vers laquelle il se déplace. Si un missile venait à entrer en collision avec une mine les deux serait détruit.



Missiles

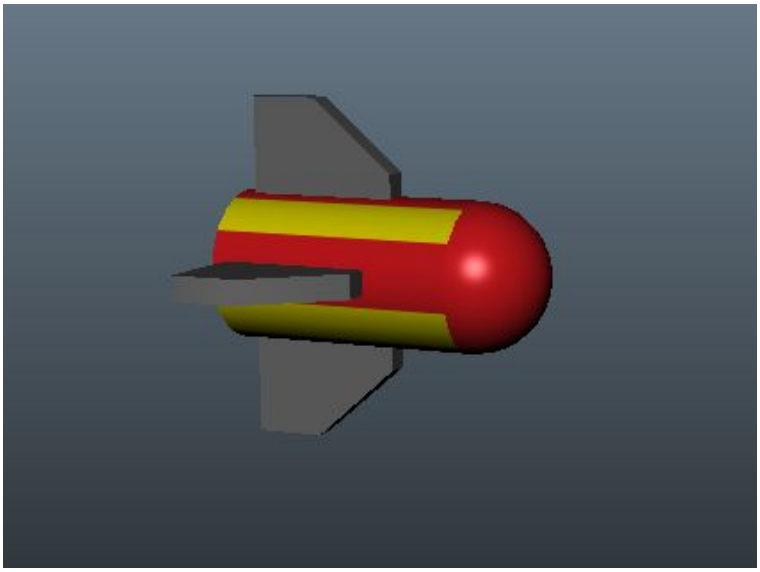
Les missile sont lancés par les chars et **apparaissent sur la carte** une fois lancés. Les missiles sont lancé dans une direction et ne peuvent pas changer de direction une fois lancé. L'utilisation de la fonction "isMissileAtPosition" dans le "World" peut vous permettre de savoir si un missile se trouve sur une case en particulier. On peut aussi avoir accès à la position du missile à partir du "Character", selon le pseudo-code suivant:

```
world.getOpponentTeam().getFirstCharacter().getMissile().getPosition()
```

Il est aussi possible d'avoir accès à la direction du missile, selon le pseudo-code suivant:

```
world.getOpponentTeam().getFirstCharacter().getMissile().getDirection()
```

Un seul missile peut être lancé par char d'assaut. Un char d'assaut pourra lancer un autre missile seulement lorsque le dernier missile qu'il a lancé est détruit. Si un missile venait à entrer en collision avec une mine les deux serait détruit. Un missile se **déplace deux fois plus rapidement** qu'un char d'assaut.



Action possible pour l'I.A.

La fonction "tick()" de votre I.A. va être appelé toutes les 60ms. La fonction "tick()" est sur un seul thread, alors plus sont exécution prend du temps plus votre I.A. va prendre du temps pour envoyer ses ordres. Les ordres sont envoyer chaque fois que la fonction "tick()" termine.

Déplacement

L'ordre de déplacement est donné à un char d'assaut avec la position vers laquelle celui-ci doit se déplacer. C'est le "GameClient" qui s'occupe de trouver le chemin nécessaire pour se rendre à une case, alors il n'est **pas nécessaire de coder un algorithme de "Path Finding".** Si la case de destination n'était pas accessible, votre char d'assaut essayerait continuellement de trouver un autre chemin pour s'y rendre.
Exemple d'utilisation:
Python:

```
aCharacter.goTo(Vector2(6,7))
```

Java:

```
character.move(new Point(5,5));
```

Tir d'un missile

L'ordre de tiré est donné à un char d'assaut avec une direction. Le char va tirer dès qu'il reçoit l'ordre peu importe sa position actuelle. S'il se trouve entre 2 cases celui-ci va attendre d'être arrivé sur sa case de destination avant de tirer. Le char restera **immobile durant une seconde** après avoir tiré. Si son tire atteint une cible sur une case adjacente à la sienne, celui-ci va **perdre aussi 1 point de vie**.
Python:

```
aCharacter.shootMissile(Direction.UP)
```

Java:

```
character.shootMissile(Missile.Direction.UP);
```

Déposer une mine

L'ordre de déposer une mine est donné à un char d'assaut. Le char va déposer sa mine dès qu'il reçoit l'ordre peu importe sa position actuelle. S'il se trouve entre 2 cases celui-ci va attendre d'être arrivé sur sa case de destination avant de déposer la mine.
Python:

```
aCharacter.dropMine()
```

Java:

```
character.dropMine();
```

Implémentation du client

Comme expliqué précédemment vos ordres seront envoyés lorsque la fonction “tick()” aura terminé sont exécution.
Un exemple d’implémentation d’I.A. est à votre disposition.
Python:

AIClient_Python/src/aiclient/AI.py

Java:

AIClient_Java/src/aiclient/AI.java

Changer les noms

Pour changer le nom de votre équipe et celui de vos char, il vous suffit de changer le contenue de la fonction “setNames” de votre ai. Un exemple est donner avec l’AI de base.

Détails techniques

Le jeux est composé de deux modules. Le premier étant le module “GameClient” qui effectue tous les calcules et qui gère le déroulement d’une partie. Il est en quelque sorte le serveur sur lequel les modules d’I.A. vont devoir se connecter. La partie commence lorsque deux modules d’I.A. sont connecté au “GameClient”. Les deux module d’I.A. qui se connectent au “GameClient” ne sont pas obligatoirement fait avec le même langages, la partie est donc complètement indépendante du langage dans lequel sont programmé les I.A.

GameClient

Le GameClient doit être partie avant les module d’I.A. pour que la partie commence. Un script est présent dans le répertoire de la compétition pour partir le “GameClient”:

```
./start.sh
```

Navigation dans l’interface

Il est possible de naviguer dans l’interface 3D à l’aide des touches “wasd”. Il est possible d'entrer dans un mode de vue à la première personne est gardant le clique de la souris enfoncé.

AIClient

Le “AIClient” va se connecter au “GameClient” un fois celui-ci partie. Pour qu’une partie commence deux “AIClient” doivent être partie un à la suite de l’autre.

Java

Un script est présent dans le répertoire de la compétition pour partir le client java:

```
./start_java.sh
```

Python

Un script est présent dans le répertoire de la compétition pour partir le client python:

```
./start_python.sh
```

Documentation

Toutes la documentation nécessaire au développement de votre I.A. est fournis avec le code du client.

Pour Python:

```
/AIClient_Python/docs/index.html
```

Pour Java:

```
/AIClient_Java/docs/index.html
```

Correction

Lors de la correction le code de chacun des participants sera révisé pour s'assurer que tous les participants ont bien suivi les directives de développement. Seule les fichier contenues dans le dossier "aiclient" seront pris en compte. Votre I.A. jouera contre celle des autres équipes pour faire un premier classement par point. Par la suite une série éliminatoire aura lieu pour découvrir qu'elle I.A. est supérieur aux autres.

Conclusion

Bonne chance à tous.
GLHF!