Projet bataille IA

Partie 2 : une vraie IA

Le but de cette seconde partie est d'améliorer l'IA de nos unités. Nous allons maintenant représenter l'IA par un arbre de décision.

Un arbre de décision contient des noeuds internes et des feuilles, comme vu en TP.
Les noeuds internes correspondent ici à des décisions, et les feuilles, à des actions.

I - Informations et Données

Pour pouvoir créer une IA plus complexe, nous avons besoin de récupérer des informations plus complexes également.

Nous allons maintenant utiliser un peu plus le potentiel de notre concept.

Nous avons quatre types de données :

- des valeurs
- des points 2D
- des unités
- des ensembles d'unités

Le programme fourni à l'IA 3 données :

- une unité (celle qui applique l'IA)
- deux ensembles d'unités (l'armée de l'unité et celle adverse)

L'arbre de décisions utilise des valeurs pour les comparaisons, et les actions sont faites par rapport à des unités ou des positions.

On peut obtenir certains types de données à partir de certains autres : par exemple, jusqu'ici nous pouvions, dans l'équipe adverse (donc un ensemble d'unités), récupérer l'unité aillant la plus faible ou la plus grande capacité indiquée. Nous appellerons cela une extraction de donnée.

Vous trouverez ci-dessous la liste des extractions a pouvoir effectuer, avec en gras, le code correspondant :

valeurs :

- direct **[val]**
- valeur de capacité d'une unité **C0-C6<unit>**
- distance d'une unité a un point **D<unit><point>**
- max/min/valeur moyenne de capacité d'un ensemble **M/m/a0-6<set>**
- distance minimale/maximale/moyenne d'un ensemble a un point **M/m/aD<set><point>**

unité :

- l'unité propriétaire de l'ia **U**
- unité qui a la valeur min/max de capacité au sein d'un ensemble **L0-L6<set> H0-H6<set>**
- unité la plus proche/loin d'un point au sein d'un ensemble **LD-HD<set><point>**

point :

- barycentre des positions d'un ensemble **B<set>**
- position d'une unité **P<unit>**

ensemble d'unités:

- armée de l'ia **A**
- armée opposée **O**
- N ayant la plus grande/petite capacité au sein d'un ensemble **NL0-6[val]<set> NH0-6[val]<set>**
- N plus proches/loin d'un point au sein d'un ensemble **NLD[val]<set> NHD[val]<set>**
- sous ensemble ayant capacité/distance >/< seuil **TL0-6[val]<set> TH0-6[val]<set>**

Codage de l'arbre complet:
L'arbre de décision complet se fait en parcours profondeur préfixe : on code un noeud, puis ses fils, si il y en a. Les noeuds de décision sont codés sous la forme suivante :

?<extraction de valeur><operateur><extraction de valeur>

Les noeuds d'action sont codés sous la forme suivante :

!<code action><extraction de parametres de l'action>

Les actions sont les suivantes :
Move : va vers un point **M<point>**
Escape :fuit un point **E<point>**
Attack : si chargé, attaque si tu peux, sinon approche toi a portée de tir **A<unit>**
Nothing : ne fait rien **N**

Exemple complet :

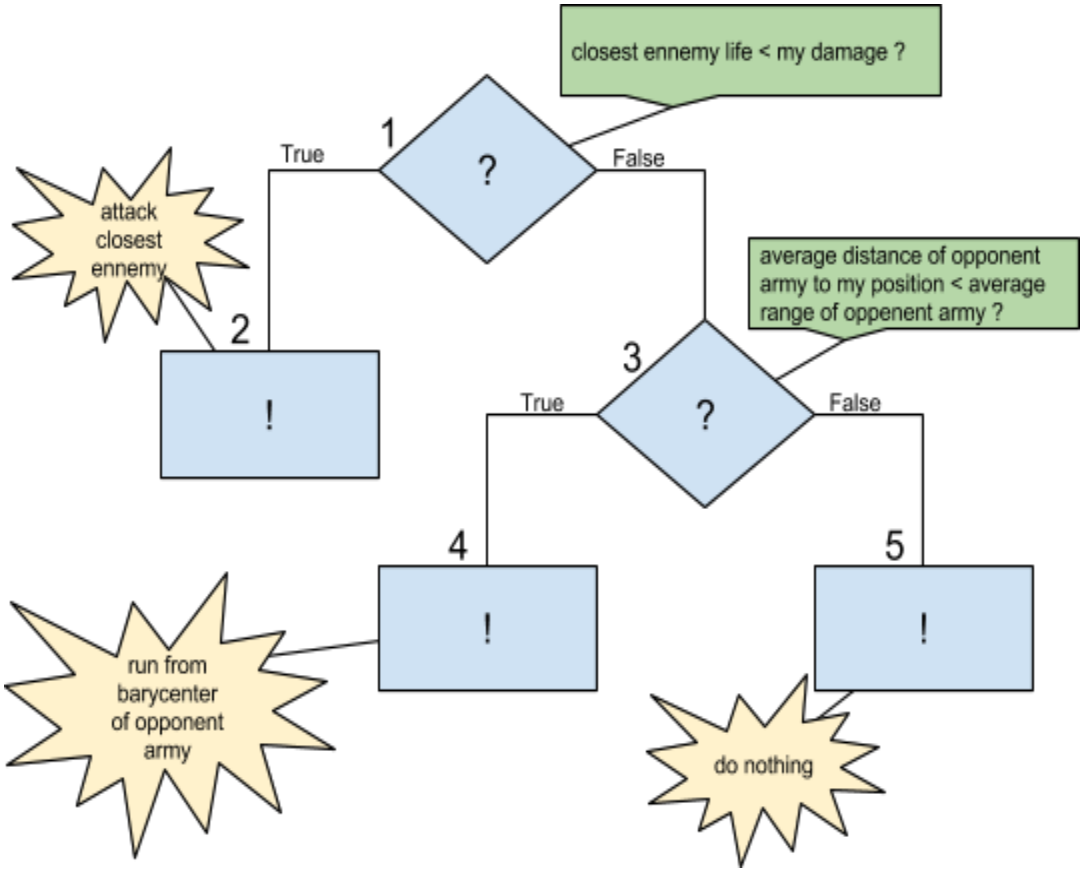

Fig. 1 Exemple d'arbre de décision avec des extractions d'information.

Dans l'exemple ci-dessus, les noeuds seront codés dans l'ordre suivant : 1, 2, 3, 4, 5
Nous avons besoin d'extraire les informations suivantes (code entre parenthèses, les espaces ne sont la que pour la compréhension):

1. la valeur de la capacité *vie* de l'unité de l'armée adverse la plus proche de l'unité courante (C1 LD O P U) **ET** la valeur de la capacité *dommages* de l'unité courante (C4 U). Le code de ce noeud est donc : **?C1LDOPU<C4U**
2. l'unité de l'armée adverse la plus proche de la position de l'unité courante (LD O P U). Le code de ce noeud est donc : **!ALDOPU**
3. la distance moyenne des unités de l'armée adverse à la position de l'unité courante (aD O P U) **ET** la valeur moyenne de la capacité *portée* des unités de l'armée adverse (a5 O). Le code de ce noeud est donc : **?aDOPU<a5O**
4. le barycentre des positions des unités de l'armée adverse (B O). Le code de ce noeud est donc : **!EBO**
5. rien. Le code de ce noeud est donc : **!N**

L'IA représentée sur la figure 1 à donc pour code:

?C1LDOPU<C4U!ALDOPU?aDOPU<a5O!EBO!N

II - Ce qui vous est demandé

Il y a deux étapes distinctes dans ce projet :

- une étape de design
- et une étape de programmation

L'étape de design consistera a trouver des solutions aux problématiques soulevées par cette façon de concevoir l'IA. Vous devrez, pour chaque problématique, décrire dans le détail une solution.
Vous consignerez cela dans un **rapport** et **présenterez devant la classe** vos solutions. La date sera indiquée dans les étapes sur MyGes. **Cela comptera pour une partie de votre note de CC.**

L'étape de programmation consistera à intégrer cette nouvelle IA dans votre application déjà existante. Vous pourrez utiliser vos solutions, ou celles d'autres groupes, si leur présentation vous a convaincu. **Votre évaluation finale sera sur l'implémentation que vous en ferez**, sur les critères habituels de qualité du code et d'utilisation convenable des outils mis a votre disposition par le C++.
Le projet sera a rendre sous forme d'une archive contenant:
- les sources,
- un exécutable windows si vous y arrivez.
- un README indiquant ce qui à été fait, par qui, et comment utiliser le programme

L'archive sera de format .zip et contiendra le nom des membres du groupe, dans l'ordre alphabétique.

III - Les problématiques

1 - Gérer les extractions

Les extractions se feront au moyen de classes spéciales, appelées extracteurs.

Un extracteur, quel qu'il soit, doit avoir accès aux trois entités passées a l'IA :

- unité courante
- armée alliée
- armée adverse

Il existe une sous classe d'extracteur par type de données, chacune extrayant un élément du type associé.

Un extracteur peut lui même contenir des extracteurs. Par exemple, l'extracteur permettant d'extraire l'unité d'un ensemble la plus proche d'un point, contiendra en attributs un extracteur d'ensemble d'unité et un extracteur de point.

Proposez l'architecture adéquate des extracteurs, en vous aidant des questions suivantes :

1. quelles sont les classes impliquées ?
2. quels sont les classes templatisées ?
3. quels sont les liens hierarchiques ?
4. comment a-t-on accès aux trois entités passées a l'IA ?
5. comment utilise-t-on nos extracteurs (cas concret) ?
6. quelles méthodes doivent-ils contenir?

2 - Gérer l'arbre de décision

Proposez l'architecture pour représenter l'IA par arbre de décisions (basez vous sur le TP, pour la structure d'arbre), qui répond aux questions suivantes :

- que retourne un noeud ?
- comment transmettre les entités passées à l'IA aux noeuds?
- comment construire l'arbre ?
- l'IA est elle partagée par tout le monde, ou chaque unité possède la sienne?
- comment recopier un arbre?

3 - Construire un arbre a partir d'un code, et inversement

Proposez des mécanismes (fonctions, classes, algorithmes, …) pour :

1. construire un extracteur à partir de son code
2. construire un arbre de décision à partir de son code
3. générer le code correspondant à un extracteur
4. générer le code correspondant à un arbre de décision

4 - Génération aléatoire

Proposez un mécanisme pour générer aléatoirement une IA.
A votre avis, vaut il mieux :

1. générer aléatoirement l'arbre de décision ? Si oui, par quel moyen?
2. générer aléatoirement un code, et de la, construire l'IA correspondante?

Justifiez votre réponse.

5 - Gérer les sous ensembles d'unités

Proposez une architecture qui permet de gérer les sous-ensemble d'armées, qui évite les recopies d'unités. Vous préciserez les classes impliquées, et leurs relations hiérarchiques, si il y en a.

6 - Croisements et Mutations

Proposez des façons de :

- croiser deux arbres de décisions
- muter un arbre de décision

Dans les deux cas, vous décrirez la façon de mettre en oeuvre vos solutions.