Projet bataille IA

Le but de ce projet est de faire s'affronter des armées d'unités, qui évolueront au fil de temps via un algorithme génétique pour devenir de plus en plus efficaces.

Cet énoncé décrit :

A. Les principales classes à créer pour réaliser le projet
B. Le déroulement d'une bataille entre deux armées
C. Comment faire évoluer les armées par croisement génétique.

Notation :
Implémenter correctement les classes énoncées dans A donne **9 points.**
Implémenter le déroulement de bataille enoncé dans B donne **1 point.**
Implémenter le processus d'algorithme génétique énoncé dans C donne **5 points.**
Les points restants sont donnés en appréciation de

- la qualité du code,
- du respect des conventions de codage
- de la qualité des commentaires
- de la bonne répartition du code en fichiers
- de la qualité de la présentation du livrable

A - Armée, Unité, Capacité et IA

Chaque armée est composée d'unités, chacune d'entre elles étant composée de capacités et d'une IA.

A1 - Capacités

Les capacités d'une unité sont les suivantes :
0- vitesse : valeur correspondant au nombre de pixels parcourus au max pendant un tour
1- points de vie : nombre de points de dégats que peut encaisser une unité avant de mourir
2- armure : nombre de points de dégats absorbés par l'armure à chaque attaque
3- régénération : nombre de points de vie régénérée à chaque tour
4- dommages de l'arme : nombre de points de dégats donnés à chaque attaque
5- portée de l'arme : on ne peut attaquer une cible plus éloignée que cette distance
6- vitesse de l'arme : nombre de tours entre deux tirs

Chaque capacité peut-être augmentée (méthode **upgrade()**), diminuée (méthode **downgrade()**), et peut fournir son niveau/nombre d'augmentations (méthode **int getLevel()**), et sa valeur (méthode **float getValue()**).

Calcul de la valeur en fonction du niveau :
Vitesse : valeur = 1+niveau
Points de vie : valeur = (1+niveau)*10
Armure: valeur = niveau*2
Régénération: valeur = niveau*3
Dommages : valeur = (1+niveau)*1.5
Portée: valeur = 10+niveau*2
Vitesse rechargement : valeur = 1000/(niveau+1)
Le niveau initial de chaque capacité est 0.

Les capacités 0,2,3,4,5 sont de simples conteneurs de valeur fixe (la valeur ne changera qu'en cas d'upgrade ou de downgrade).

La capacité 1 (points de vie) est particulière : sa valeur est le nombre de points de vie actuel de l'unité. Il faudra donc garder en mémoire son nombre de points de vie maximum. La valeur est donc initialisée au nombre de points de vie max, diminue a chaque dommage pris, et augmente en fonction de la capacité de régénération (sans pouvoir dépasser la valeur maximale pour autant).

La capacité 6 (vitesse rechargement) est elle aussi particulière : sa valeur est le nombre de tours à attendre avant de tirer. Il faudra donc garder en mémoire le temps de recharge maximal. La valeur est initialisée à 0, est mise a sa valeur maximale a chaque tir, et est décrémentée à chaque tour jusqu'a 0.

A2 - Intelligence Artificielle

L'intelligence artificielle est un foncteur qui :

- prend trois paramètres : une unité, son armée, l'armée adverse.
- retourne une action à effectuer :
    - action de type shoot : peut fournir l'id de la cible sur laquelle tirer
    - action de type move : peut fournir la position vers laquelle se déplacer

Il vous est demander de pouvoir créer l'IA suivante :

- si on ne peut pas tirer, il faut se mettre hors de portée des ennemis
- si on peut tirer on choisit sa cible
    - si la cible est a portée, on tir dessus
    - sinon, on se rapproche de la cible

Ce qui va “customiser” votre IA est le choix de la cible : pour toutes les capacités, ainsi que la distance à l’unité courante on pourra choisir de prendre pour cible l’unité ennemie ayant la plus grande ou la plus petite valeur.

Chaque IA possible grâce à ces combinaisons se verra attribuer un code de deux caractères:

- la première correspondant à si on choisi la cible avec la plus grande valeur (“H”) ou la plus petite (“L”)
- la seconde correspond a l’indice de la capacité choisie, ou à D dans le cas ou on choisi la distance comme critère.

Des exemples de ciblages possibles (avec leur code entre parenthèse) :

- l’ennemi le plus proche (***LD***)
- l’ennemi qui a le plus de tours a attendre avant son prochain tir (***H6***)
- l’ennemi qui a le moins de vie ***(L1***)
- l’ennemi qui peut faire le plus de dommages (***H4***)

A3 - Unité

Une unité peut fournir :

- un identifiant unique (**int getId()**)
- un accès à chaque capacité via son nom (méthodes **SpeedCapacity& getSpeed()**, **HealthCapacity& getHealth()**, etc…)
- un accès à chaque capacité via son index 0 -> vitesse, 1 -> points de vie, etc... (opérateur [])
- son niveau global (méthode **int getLevel()**)
- sa position (méthode **Point getPosition()**)

Les actions d’une unité sont :

- rafraichir ses informations (méthode void **refresh()**)
    - mettre la valeur de la vie a *max(valeur max de vie, valeur de vie + regeneration)*
    - décrémenter le temps de latence avant son prochain tir
- changer de position (méthode **void setPosition()**)
- tirer (méthode **bool shoot()**) :
    - si temps de latence est à 0, mettre temps de latence au maximum et retourne vrai
    - sinon retourne faux
- prendre des dommages (méthode **void takeDamage(float value)**) :
    - diminuer la valeur de la vie de (**value** - valeur armure)
- indiquer qu’elle est encore en vie (méthode **bool isAlive()**):
    - si la valeur de la vie est inférieure ou égale à 0, retourner faux
    - sinon retourner vrai

Les constructeurs demandés pour une unité sont les suivants :

- un seul paramètre de type int, représentant le niveau global de l'unité, les niveaux seront alors répartis aléatoirement entre les capacités, et on attribuera une IA aléatoirement.
- un code IA, et un paramètre entier pour chaque capacité, correspondant au niveau de celle ci.

A4 - Armée

Une armée peut fournir :

- la liste de ses unités (méthode **std::vector<Unit*> getUnitsList()**)
- une de ses unités par son id (**Unit& getUnit(int id)**)
- son nombre d'unités (**int size()**)
- son unité la plus proche d'un point donné (**Unit& getNearestUnit(const Point& p)**)
- son unité la plus éloignée d'un point donné (**Unit& getFurtherUnit(const Point& p)**)
- celle a la valeur de capacité la plus basse (**Unit& getLowestUnit(int capa_index)**)
- celle a la valeur de capacité la plus haute (**Unit& getHiggestUnit(int capa_index)**)

Ses actions sont :

- purger les unités mortes (**void purge()**)

Deux constructeurs sont requis :

- un prenant un nombre d'unités, et le niveau global (identique) pour chacune d'elle. Dans ces cas la, on générera aléatoirement les Unités grâce au constructeur dédié.
- un prenant un vecteur d'Unités, qui sera recopié.

B - Déroulement d'une confrontation d'armées

Au départ on a nos deux armées.
Tant que les deux armées ont au moins une unité :

- on récupère la liste d'unités de chaque armée
- pour chaque unité, prise de façon aléatoire :
    - on lance son IA
    - on applique l'action demandée en retour de l'IA
    - on lance la purge de l'armée adverse

Le score de l'armée perdante est 0.
Le score de l'armée gagnante est le nombre d'unités à la fin de la confrontation.

Un affichage décrivant ce qu'il se passe doit être effectué à chaque tour, par exemple en utilisant la syntaxe suivante :

```
============= Tour 14 =================
Unité 23 (Armée A)  attaque Unité 3 (Armée B) qui n'a plus que 54 HP
Unité 2 (Armée B) bouge en position (10,562)
```

Les scores finaux doivent s'afficher également sur la sortie standard.

C - Déroulement d'une session complète

1. L'utilisateur choisi 5 valeurs entières **I,T, N, X**, et **Y**.
2. On génère aléatoirement **N** armées de **X** unités avec chacune un niveau global de **Y**.
3. On fait **I** fois :
    a. On fait combattre chaque armée contre les **N-1** autres (en créant des copies), et on additionne les scores obtenus.
    b. On classe les armées par score décroissant (la première est donc celle de score maximal)
    c. Si le score de la première armée est supérieur à **T**, on sort de la boucle
    d. On créé notre nouvelle génération d'armées (qui remplacera la génération courante) de la façon suivante :
        i. on garde les **(N*0.1)** meilleures armées
        ii. on prend un croisement issu de chacune des **(N*0.3)** meilleures armées avec une autre prise aléatoirement
        iii. on prend une mutation de chacune des **(N*0.3)** meilleures armées
        iv. on génère **(N*0.3)** nouvelles armées aléatoirement
4. En sortant de la boucle, on a normalement une armée meilleure que les autres, il ne nous reste plus qu'a la sauvegarder.

On notera que T doit être inférieur à **(N-1)*X** qui est le score maximal possible (une armée gagne tous ses affrontements sans perdre une seule unité).

Ci dessous, le détail des différentes méthodes à ajouter pour effectuer facilement une session.

Mutation d'une unité :
une unité peut être amenée à muter (méthode **Unit mutate()**). Cela consiste a diminuer une capacité qui n'est pas déjà au niveau minimal, et à augmenter une autre.

Croisement de deux unités :
Deux unités peuvent être croisées, pour générer une nouvelle unité (opérateur *****).
Le but est d'obtenir une nouvelle unité dont:

- le niveau global est compris dans l'intervalle des niveaux globaux des deux parents
- le niveau de chaque capacité est inférieur ou égal au maximum de celui de chaque parent
- l'IA est l'une des deux parents

Mutation d'une armée :

Une mutation d'une armée (méthode **Army mutate()**) peut-être effectuée de plusieurs façons (libre à vous de choisir l'une d'entre elles, ou de tirer aléatoirement le type de mutations)

- en remplaçant une unité par une nouvelle, générée aléatoirement
- en faisant muter un sous ensemble aléatoire des unités

Croisement de deux armées :

Le croisement de deux armées (opérateur *) est effectué de la façon suivante, en prenant, dans les proportions de votre choix :

- des unités de l'armée A
- des unités de l'armée B
- un croisement d'une unité de l'armée A avec une unité de l'armée B

Sauvegarde d'une armée :

Le nom du fichier de sauvegarde sera ***army_X_Y.save*** avec X le nombre d'unités dans l'armée, et Y son niveau global.

Le fichier sera composé d'un nombre de lignes égal au nombres d'unités, et chaque ligne sera composée de 7 nombres séparés par des espaces, correspondant aux niveaux des 7 capacités, suivis d'un espace et de l'identifiant de l'IA utilisée par cette unité.

Par exemple, pour une unité qui à respectivement les niveaux 10,2,4,0,1,15,8 pour les capacités "vitesse", "points de vie", "armure", "régénération", "dommages", "portée", "vitesse de tir", et qui utilise l'IA qui cible en priorité l'ennemi le plus rapide, la ligne sera :
10 2 4 0 1 15 8 H0

Rendu :
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.