

Rapport TP2

- Paul Seichais
- Malek El Ouerghi
- Pierre Lochouarn

Exercice 1:

```
C:\Program Files\Java\jdk-8.0.121\bin>java -x
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
18/04/30 15:46:40 INFO SparkContext: Running Spark version 2.2.0
18/04/30 15:46:41 INFO SparkContext: Submitted application: BDRTPlex1
18/04/30 15:46:41 INFO SecurityManager: Changing view acls to: Malek
18/04/30 15:46:41 INFO SecurityManager: Changing modify acls to: Malek
18/04/30 15:46:41 INFO SecurityManager: Changing view acls groups to:
18/04/30 15:46:41 INFO SecurityManager: Changing modify acls groups to:
18/04/30 15:46:41 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(Malek); groups with view permissions: Set(); users with modify permissions: Set(Malek); groups with modify permissions: Set()
18/04/30 15:46:42 INFO Utils: Successfully started service 'sparkDriver' on port 14144.
18/04/30 15:46:42 INFO SparkEnv: Registering MapOutputTracker
18/04/30 15:46:42 INFO SparkEnv: Registering BlockManagerMaster
18/04/30 15:46:42 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
18/04/30 15:46:42 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
18/04/30 15:46:42 INFO DiskBlockManager: Created local directory at C:\Users\wash1\AppData\Local\Temp\blockmgr-9f525427-d9ad-465d-b6e4-3b5144ff9aa8
18/04/30 15:46:42 INFO MemoryStore: MemoryStore started with capacity 897.6 MB
18/04/30 15:46:42 INFO SparkEnv: Registering OutputCommitCoordinator
18/04/30 15:46:42 INFO Utils: Successfully started service 'SparkUI' on port 4040.
18/04/30 15:46:42 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://192.168.0.129:4040
18/04/30 15:46:42 INFO Executor: Starting executor ID driver on host localhost
18/04/30 15:46:42 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 14187.
18/04/30 15:46:42 INFO NettyBlockTransferService: Server created on 192.168.0.129:14187
18/04/30 15:46:42 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
18/04/30 15:46:42 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 192.168.0.129, 14187, None)
18/04/30 15:46:42 INFO BlockManagerMasterEndpoint: Registering block manager 192.168.0.129:14187 with 897.6 MB RAM, BlockManagerId(driver, 192.168.0.129, 14187, None)
18/04/30 15:46:42 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 192.168.0.129, 14187, None)
18/04/30 15:46:42 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 192.168.0.129, 14187, None)
Fetching Monsters list...
Monsters retrieved : 1363
Fetching spells for monsters...
Spells fetched !
Process finished with exit code 0
```

Exécution du crawler pour retrouver les monstre par sort.

Pour cela nous avons réaliser un programme en scala ou nous envoyons des requêtes sur un site et parsons ensuite le résultat.

Resultat : <https://gist.github.com/washigeri/7fdd1a85b66f0cb7eef0d2b2511d6675>

Exercice 2 :

Question ouverte : Comment gérer efficacement un système de distance 3D avec un graphe d'agents distribué? Avoir un système de distance 3D permet d'avoir accès aux attaques de zone, le vol, les arcs et flèches etc.

On met la distance sur le sommet? sur l'arête? problèmes de collision etc.

Nous avons choisi de stocker les positions (x,y) de chaque sommet du graphe. La distance en elle-même n'est stockée nulle part, elle est calculée (distance euclidienne) sur le moment si on en a besoin. Ceci nous permet de modifier facilement les positions de chacun sans toucher à la structure du graphe et sans toucher aux arêtes. Pour gérer facilement les collisions, on peut donner une hitbox sphérique a chaque sommet et ainsi pour vérifier la collision entre deux sommet, il suffit de regarder si la distance entre les deux sommets est inférieure à la somme des rayons des deux sphères de collisions.

Combat 1 :

Nous avons donc pour ce combat créé un graphique où tous les monstres sont reliés au solar.

Nous avons donc le Solar, 10 Worgriders, 4 Barbarian Orcs et 1 warlords

Après déroulement du combat voici le résultat final :

```
-----COMBAT 1-----
-----FIN-----
ETAT APRES LE COMBAT
SURVIVANTS
Solar 1 HP: 363
-----FIN COMBAT 1-----
```

Pour arriver à ce résultat voici notre workflow :

- Chaque noeud représente un monstre et chaque arête représente un type de relation entre les différents noeuds (dans ce combat il n'y a que des arêtes de type ENEMY)
- Chaque monstre dispose d'une méthode action ainsi que d'une position représentée selon des axes X, Y
- Egalement il dispose chacun d'une méthode damage qui renvoie des dégâts en fonction de la classe du monstre
- Nous lançons notre méthode d'exécution qui :
 - Créer une liste d'action possible pour chaque monstre en fonction des différentes arêtes qui lui sont liées ainsi que la distance entre chaque monstre.
 - Puis un joint vertex qui choisit l'action que le noeud effectuera et sur quel cible cela sera appliquée
 - Ensuite nous relançons un aggregate pour envoyer les messages d'attaque au noeud ciblé ou, dans le cas d'un déplacement ceux-ci sont renvoyés sur le noeud actuel
 - Enfin le dernier joint vertex applique les actions qui ont été envoyées précédemment

```
-----COMBAT 1-----
-----DEBUT DU TOUR-----
Tour 1
Warlord16 avance vers Solar1
Worgrider4 avance vers Solar1
BarbarianOrc12 avance vers Solar1
Worgrider11 avance vers Solar1
Worgrider3 se fait attaquer a RANGED de Solar pour 21 points de degats
Worgrider3 avance vers Solar1
-----MONSTER-----
Worgrider 3 est mort
-----
```

Pour choisir les actions tous les monstres choisissent les attaques en priorité, par exemple si le solar a le choix entre un mouvement ou une attaque à distance il choisira d'attaquer à distance, cependant en cas de possibilité d'attaquer au corps à corps, ils préféreront l'attaque corps à corps.

Combat 2 :

Le déroulement du combat 2 est exactement le même que le combat 1, sachant que le dragon est pris en compte, celui-ci adopte une stratégie assez basique, il tente de tuer le solar dès le début du combat en traversant le champs de bataille grâce à l'alterSelf, et si cela échoue il s'envole et bombarde les anges de façon aléatoire.

Pour ce programme nous avons nos ordinateurs en cluster afin d'exécuter plus rapidement.

Voici le résultat final :

```
-----MONSTER-----  
-----FIN-----  
  
ETAT APRES LE COMBAT  
SURVIVANTS  
Astral 8 HP: 172  
Solar 1 HP: 363  
Astral 9 HP: 172  
Astral 10 HP: 172  
Planetar 2 HP: 229  
Planetar 3 HP: 229  
MovanicDeva 4 HP: 117  
MovanicDeva 5 HP: 126  
Astral 6 HP: 172  
Astral 7 HP: 172  
----- FIN COMBAT 2-----
```

Exécution en cluster (2 PCs) : <https://youtu.be/XLjVecyszXQ>

En cluster, on n'a pas les print qui détaillent l'avancement du combat (attaque, bouge, soigne, tire à distance, etc.) vu que les print se font sur chaque worker.