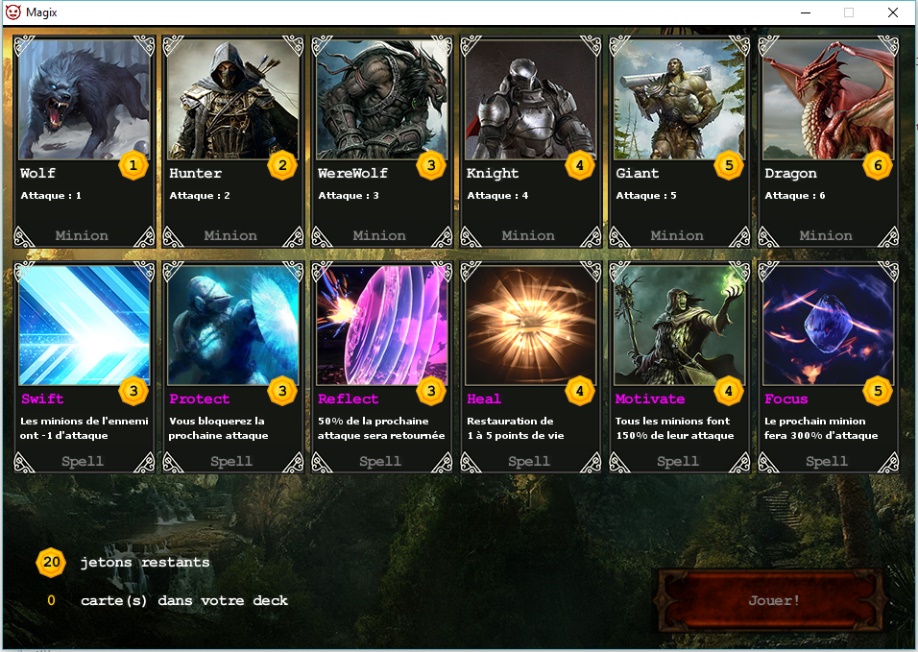
# Intégration de MongoDB pour analyser l’état d’un jeu (BI)

Ce jeu de cartes vient d’être conçu (*mettons*). Malheureusement pour eux, le développeur ainsi que le créateur n’ont pas les moyens de se payer une équipe de testeurs afin de voir si les cartes sont bien équilibrées. Il vous demande donc d’analyser des exemples de parties afin de constater l’état du jeu.

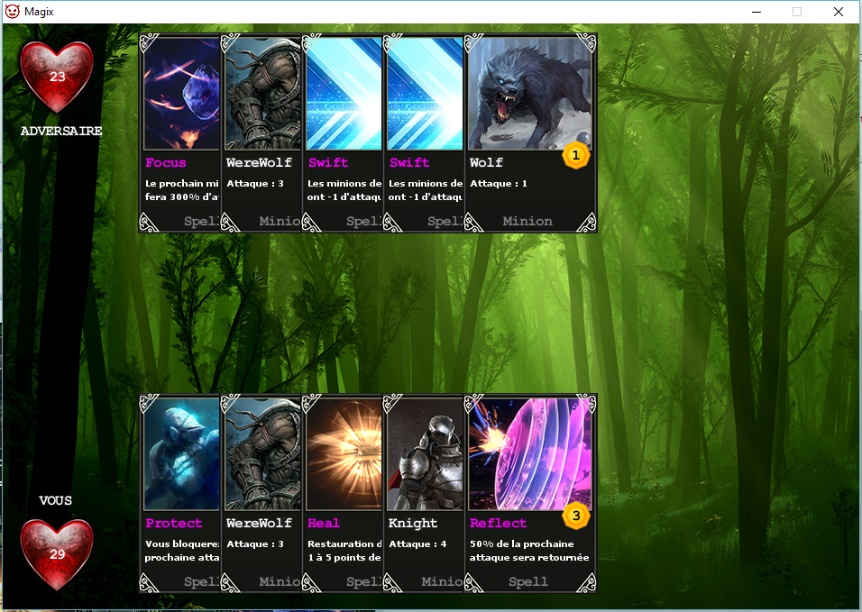


## Le jeu « Magix »

Ce jeu consiste à choisir quelques cartes qui, lors de la partie, seront déposées aléatoirement sur le plateau de jeu. Le but est d’infliger le plus de points à son adversaire.

Il y a deux conditions à la fin d’une partie :

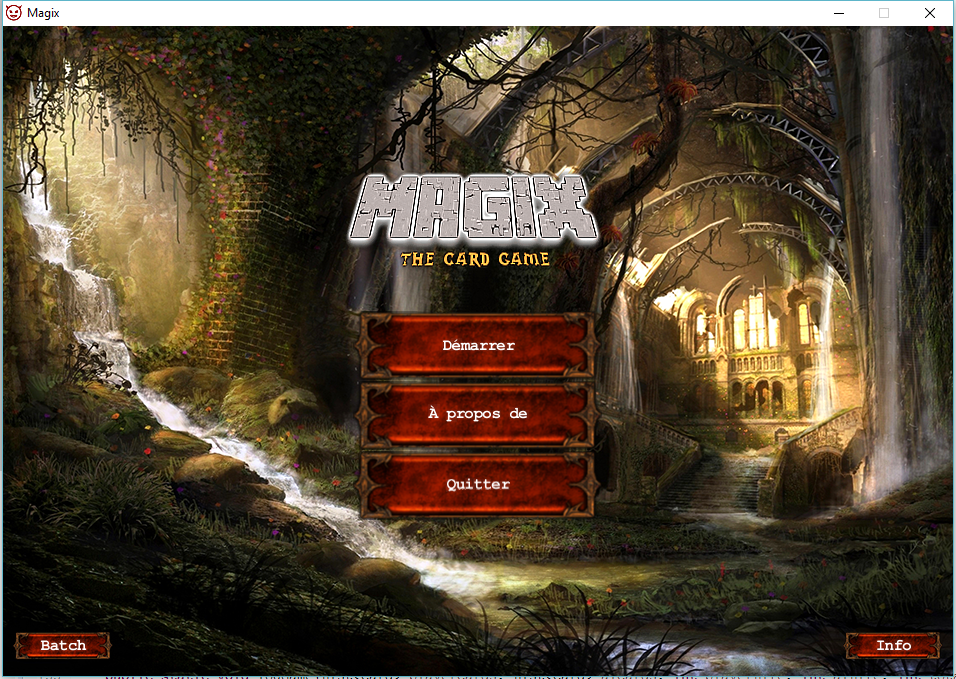
1. L’un des deux joueurs n’a plus de vie (<= 0).
2. Il n’y a plus de carte à jouer. Dans ce cas, celui qui possède le plus de points de vie gagne.



## Notes

* Pour réaliser ce projet, je vous recommande FORTEMENT de suivre les étapes en ordre.
* IntelliJ a été utilisé pour créer le programme, vous devriez donc l’utiliser pour compléter ce travail.
* Si vous avez des questions, n’hésitez pas à me demander de l’aide.
* **IMPORTANT** : Ne modifiez en aucun cas la classe « CorClient.java », sous peine d’avoir 0.

## Le menu principal



Démarrer : Démarrer une partie

À propos de : Informations sur la provenance des images

Quitter : *Action* é*vidente…*

Batch : Permet de lancer des parties fictives

Info : Page contenant des informations intéressantes sur le jeu

## Exemples d’accès à MongoDB dans le programme

### Exemple 1 : Avoir accès à la collection

MongoDatabase connection = DBConnection.getConnection();

MongoCollection<Document> collection = connection.getCollection("maCollectiondExceptions");

### Exemple 2 : Insertion d’un document dans une collection

try {

MongoDatabase connection = DBConnection.getConnection();

MongoCollection<Document> collection = connection.getCollection("maCollectiondExceptions");

Document doc = new Document (); // Création d’un document

doc.append("type", "Exception");

doc.append("time", System.currentTimeMillis());

doc.append("info", "Erreur système");

// Pour ajouter une liste de chaînes de caractères dans le document

List<String> elements = new ArrayList<String>(); // Création d’une liste fictive

elements.add("Lorem");

elements.add("Ipsum");

elements.add("Sin");

doc.append("liste", elements); // Ajoute de la liste au document

collection.insertOne(doc); // Insertion dans la collection

}

catch (Exception e) {

e.printStackTrace();

}

### Exemple 3 : Parcourir une collection

MongoDatabase connection = DBConnection.getConnection();

MongoCollection<Document> collection = connection.getCollection("maCollectiondExceptions");

FindIterable<Document> iterator = collection.find();

try {

MongoCursor<Document> cursor = iterator.cursor();

while (cursor.hasNext()) {

Document document = cursor.next();

String nom = document.getString("nomDeLaColonneOuPropriete");

System.out.println(nom);

}

}

catch (Exception e) {

e.printStackTrace();

}

### Exemple 4 : Rechercher dans une collection, trier puis limiter

MongoDatabase connection = DBConnection.getConnection();

MongoCollection<Document> collection = connection.getCollection("maCollectiondExceptions");

Document query = new Document("type", "Fatal");

Document orderBy = new Document("createdInMsec", -1); // -1 = DESC, 1 = ASC

FindIterable<Document> iterator = collection.find(query).sort(orderBy).limit(10);

[…]

### Exemple 5 : Faire une fonction d’agrégation (SUM, AVG, …)

L’exemple suivant permet de faire la somme de l’attribut « temps » en groupant toute la collection « collExceptions ».

MongoDatabase connection = DBConnection.getConnection();

MongoCollection<Document> collection = connection.getCollection("maCollectiondExceptions");

// {$group : {\_id : null, total : { $sum : temps }}}

// SQL : SELECT SUM(temps) FROM collExceptions GROUP BY *null*

Document group = new Document(

"$group", new Document("\_id", null).append(

"total", new Document( "$sum", "$temps" )

)

);

List<Document> list = new ArrayList<Document>();

list.add(group);

AggregateIterable<Document> iterable = collection.aggregate(list );

MongoCursor<Document> cursor = iterable.cursor();

String sumTxt = "0";

if (cursor.hasNext()) {

sumTxt = cursor.next().getDouble("total") + "";

}

System.out.println("Temps perdu : " + sumTxt);

### Exemple 6 : Compter le nombre de fois qu’un élément se trouve dans une liste

MongoDatabase connection = DBConnection.getConnection();

MongoCollection<Document> collection = connection.getCollection("maCollectiondExceptions");

Document where = new Document("liste", "Lorem"); // WHERE Lorem IN liste

where.append("type", "Exception"); // AND type = "Exception"

long count = collection.countDocuments(where);

## Les tâches à réaliser

Ces tâches doivent être réalisées en ordre, autrement il sera difficile de tester que tout fonctionne.

### Tâche #1 : Familiarisation du jeu

Avant toute chose, vous devriez vous familiariser avec le fonctionnement du jeu. Pour une fois que vous pouvez jouer à un jeu en classe, profitez-en! (5 minutes max!)

Vous n’avez pas à aller dans la section « Info » et « Batch » pour le moment.

### Tâche #2 : Conserver les informations d’une partie

Cette tâche consiste à sauvegarder les informations d’une partie, afin de pouvoir en faire l’analyse éventuellement. Pour ce faire, il s’agit de compléter la méthode *GameDAO.logGame()*.

**IMPORTANT : Vous n’avez pas à créer la base de données (qui se nomme « magix\_db »), elle sera créée automatiquement. Le nom de votre collection est au choix.**

Vous devez sauvegarder :

* La date qu’a eu lieu la partie (via la fonction : System.currentTimeMillis() )
* Le nombre restant de points de vie du joueur
* Le nombre restant de points de vie de l’IA (intelligence artificielle)
* Le nombre de « rounds » (de tours joués, ou cartes déposées par un joueur) dans la partie avant la fin
* La liste de cartes jouées par le joueur (une liste contenant les noms de cartes (i.e. getName())
* La liste de cartes jouées par l’IA (une liste contenant les noms de cartes (i.e. getName())
* L’issue de la partie (qui a gagné) (le joueur, l’IA ou une partie nulle)
  + Sauvegarder l’issue comme un INT, ça permet des recherches beaucoup plus rapides :
    - 0 = nulle
    - 1 = joueur
    - 2 = IA

Lorsque vous aurez terminé cette tâche, vous devriez faire l’essai d’une partie et voir si elle est bien présente dans la base de données via une console « mongo ».

#### Notes/Requête/Énoncé:

try {

MongoDatabase connection = DBConnection.getConnection();

MongoCollection<Document> collection = connection.getCollection("parties");

int resultat = 0;

List<String> nomPlayer = new ArrayList<String>();

for(int i = 0; i < playerCards.size(); i++) {

nomPlayer.add(playerCards.get(i).getName());

}

List<String> nomAi = new ArrayList<String>();

for(int i = 0; i < aiCards.size(); i++){

nomAi.add(aiCards.get(i).getName());

}

if (playerLife > aiLife) {

resultat = 1;

}

else if (playerLife < aiLife) {

resultat = 2;

}

else {

resultat = 0;

}

Document doc = new Document();

doc.append("time", System.currentTimeMillis());

doc.append("playerLife" ,playerLife);

doc.append("aiLife", aiLife);

doc.append("totalRounds", totalRounds);

doc.append("listePlayer", nomPlayer);

doc.append("listeAI", nomAi);

doc.append("gagnant", resultat);

collection.insertOne(doc);

} catch (Exception e) {

e.printStackTrace();

}

### Tâche #3 : Afficher le nombre de parties

Il s’agit de compléter la méthode *GameDAO.getGameCount()*, qui permet de retourner le nombre de parties qui ont été jouées. Ne **pas** utiliser de curseur, il y a plus simple!

Note : Cette information est affichée dans la fenêtre « Info », accessible à partir du menu principal du jeu.

#### Notes/Requête/Énoncé:

MongoDatabase connection = DBConnection.getConnection();

MongoCollection<Document> collection = connection.getCollection("parties");

long count = 0;

try {

count = collection.countDocuments();

} catch (Exception e) {

e.printStackTrace();

}

return count;

### Tâche #4 : Avoir les informations des dernières parties

Dans la fenêtre « Info », accessible du menu principal, il devrait y avoir les informations des dernières parties jouées.

L’affichage devrait ressembler à ceci :



"Gagnée par : --" signifie une partie nulle.

Exemple permettant de formater des millisecondes en date :

long time = System.currentTimeMillis();

Date date = new Date(time);

date.toString(); // Affichera un format similaire à : Web Mar 30 13:33:12 EDT 2016

Pour compléter cette tâche, il s’agit de terminer la méthode *GameDAO.* *getLatestGamesResults()*. N’oubliez pas de trier les résultats selon la date de la partie (descendant).

Note : La méthode retourne une liste de « String », où chaque « String » représente les informations d’une partie. Aussi, pour bien formater la ligne (pour que tout soit aligné), utilisez « String.format(…) ».

Note 2 : N’oubliez pas d’appliquer une **limite** de résultats, selon le paramètre de la méthode *getLatestGamesResults()*.

Par exemple :

// L’exemple suivant affiche la chaîne Salut (en prenant l’espace de 15 caractères), suivi de

// « Fred, je vais », suivi de bien (avec 7 espaces vides)

String result = String.format("%-15s Fred, je vais %-10s", "Salut", "bien");

#### Notes/Requête/Énoncé:

MongoDatabase connection = DBConnection.getConnection();

MongoCollection<Document> collection = connection.getCollection("parties");

final List<String> results = new ArrayList<String>();

Document orderBy = new Document("time", -1);

FindIterable<Document> iterator = collection.find().sort(orderBy).limit(numberOfResults);

try {

MongoCursor<Document> cursor = iterator.cursor();

while (cursor.hasNext()) {

Document doc = cursor.next();

// Temps

long time = doc.getLong("time");

Date date = new Date(time);

String formattedDate = date.toString();

// Gagnant

int gagnant = doc.getInteger("gagnant");

String formattedGagnant = "";

if (gagnant == 1) {

formattedGagnant = "Joueur";

}

else if (gagnant == 2) {

formattedGagnant = "IA";

}

else {

formattedGagnant = "--";

}

// Rounds

int rounds = doc.getInteger("totalRounds");

// Insert

String result = String.format("%-20s\tGagné par : %-10s\t\t%-2s tours", formattedDate, formattedGagnant, rounds);

results.add(result);

} catch (Exception e) {

e.printStackTrace();

}

return results;

### Tâche #5 : Savoir quelles sont les cartes les plus utilisées dans les parties gagnantes

Afin de savoir quelles sont les cartes qui sont les plus fortes, donc qui permettent de gagner le plus souvent, vous devez compléter *GameDAO.getNumberOfWonGames(Card)*. L’idée est de calculer, pour chaque carte, les parties où elle était présente et où le vrai joueur a gagné, **pas** l’adversaire IA.

**IMPORTANT : Utilisez deux filtres (parties gagnées par l’usager et où la carte est xyz), puis utilisez la fonction count().**

Voici un exemple de résultat ou la carte Hunter a permis de gagner 5 parties, la carte Knight 4 parties, etc. :



#### Notes/Requête/Énoncé:

MongoDatabase connection = DBConnection.getConnection();

MongoCollection<Document> collection = connection.getCollection("parties");

long count = 0;

try {

String nomCarte = c.getName();

Document where = new Document("listePlayer", nomCarte);

where.append("gagnant", 1);

count = collection.countDocuments(where);

} catch (Exception e) {

e.printStackTrace();

}

return count;

### Tâche #6 : Combien de tours sont joués en moyenne lors d’une partie

En utilisant une fonction d’agrégation, combien de tours sont joués en moyenne lors des parties ?

Pour faire cette tâche, il faut compléter la méthode *GameDAO.getAverageRounds()*.

Exemple de résultat :



#### Notes/Requête/Énoncé:

MongoDatabase connection = DBConnection.getConnection();

MongoCollection<Document> collection = connection.getCollection("parties");

double avg = 0;

try {

Document group = new Document(

"$group", new Document("\_id", null).append(

"total", new Document("$avg", "$totalRounds")

)

);

List<Document> liste = new ArrayList<Document>();

liste.add(group);

AggregateIterable<Document> iterable = collection.aggregate(liste);

MongoCursor<Document> cursor = iterable.cursor();

if (cursor.hasNext()) {

avg = cursor.next().getDouble("total");

} catch (Exception e) {

e.printStackTrace();

}

}

return avg;

### Tâche 7 : Tester sur un grand nombre de parties

Avec quelques parties jouées seulement, vos requêtes sont performantes. Cependant, comment savoir si elles le sont vraiment lorsqu’elles sont exécutées une grosse base de données. Il faut donc remplir la base de données de parties fictives.

En appuyant sur le bouton « Batch », ajouter 10 parties, puis s’il n’y a pas d’erreur, 200 000 parties !

Lorsque ce sera terminé, si vous cliquez sur le bouton « Info » et que la page se charge en moins de 1 seconde, tout est OK. Sinon, il vous faudra **créer des index (via une console « mongo ») simples et possiblement composées.** Aussi, le sens des index (1 pour croissant, -1 pour décroissant), est important!

Modifiez vos requêtes et faites les index nécessaires afin que la page se charge en moins de 1 seconde.

#### Notes/Requête/Énoncé:

db.parties.createIndex({time: -1})

db.parties.createIndex({listePlayer: 1, gagnant: 1})

### Tâche 8 : Faire la correction

Il est temps d’envoyer votre programme au serveur de correction. Bonne chance!

### Tâche 9 : Analyse du nombre de points de vie (extra, si vous avez terminé)

Note : Cette tâche se fait directement dans la console « mongo », pas en Java.

Le créateur du jeu vous demande maintenant s’il y a trop de points de vie dans le jeu? Vous devez donc réfléchir, puis utiliser vos 200 000 parties fictives afin de déduire si ce serait une bonne idée de diminuer le nombre de points de vie des joueurs.

Réponse et explication :

#### Notes/Requête/Énoncé:

### Tâche 10 : Le premier joueur (extra, si vous avez terminé)

Note : Cette tâche se fait directement dans la console « mongo », pas en Java.

Présentement et pour chaque tour, l’adversaire (IA) est le premier à déposer sa carte. Est-ce que cela influence les chances de gagner de celui-ci, du joueur ou est-ce que c’est sans importance ?

Réponse et explication :

#### Notes/Requête/Énoncé: