Analyse technique du projet d’Analyse de programmation

Table des matières

Introduction

Le projet est un projet qui sera développé sous java. Il a pour but de se faire affronter 20 bots qui auront chacun une classe et des points de vies, en deux équipes de 10. Chaque équipe choisis avant le combat une stratégie à adopter. A la fin du combat, certaines données seront enregistrées dans des tables SQL afin de pouvoir faire des statistiques. La partie se termine lorsqu'une des équipes n'a plus de combattant.

Outils utilisés

IDE : Visual Studio Code

GSDB : Mysql 8.0.21(Wamp)

Langage : Java 15 / SQL

Gestion du temps : Youtrack (Outil JetBrain).

La base de données

La base de données doit respecter les 3 formes normales.

Comme chaque personnage doit avoir un nom et un prénom, il y a deux tables qui contiennent pour ces données : `[nom]` et `[prenom]`.

Ensuite une table `[classe]` qui contient les 4 classes demandé avec leur valeur max et min.

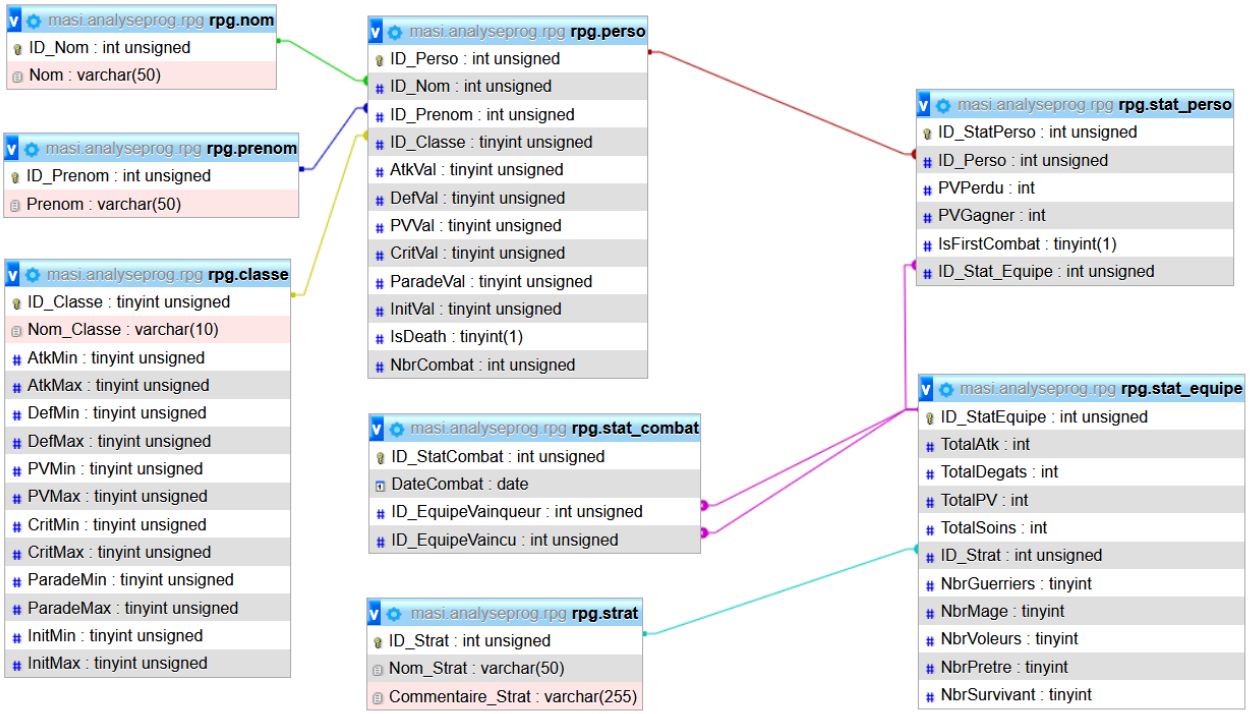
Les personnages sont créer avec la procédure stockée `[newPerso()]`. Leur données est enregistré dans la table `[perso]`.

Une vue `[combattant]` est utilisée pour avoir la liste des personnages apte au combat. Cette vue est trié de manière aléatoire afin de pouvoir sortir 20 combattants au hasard.

Pour les statistiques, trois tables sont créées : `[stat_combat]`, `[stat_equipe]` et `[stat_perso]`.

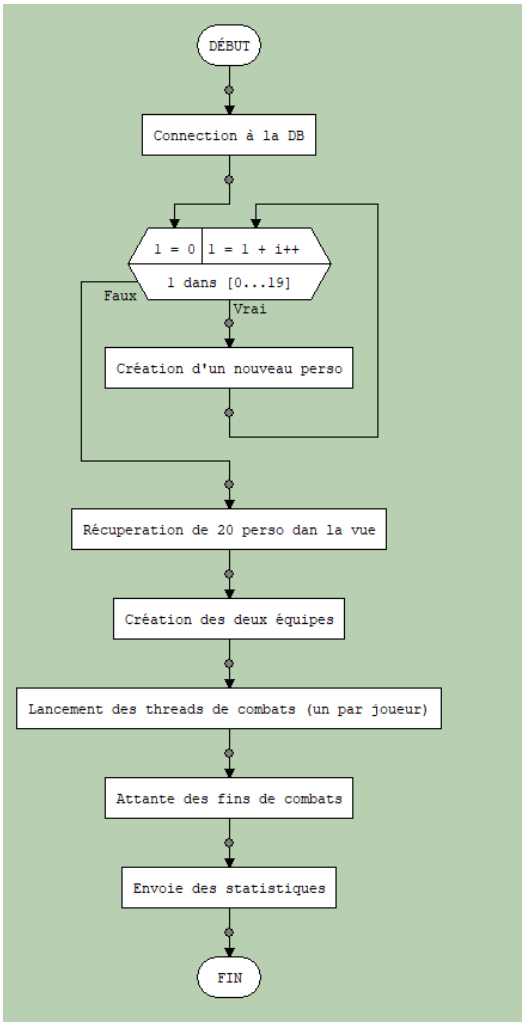
Pour finir, une table qui contiendra les algorithme de combat est aussi utilisée : `[strat]`

Schéma de la base de données

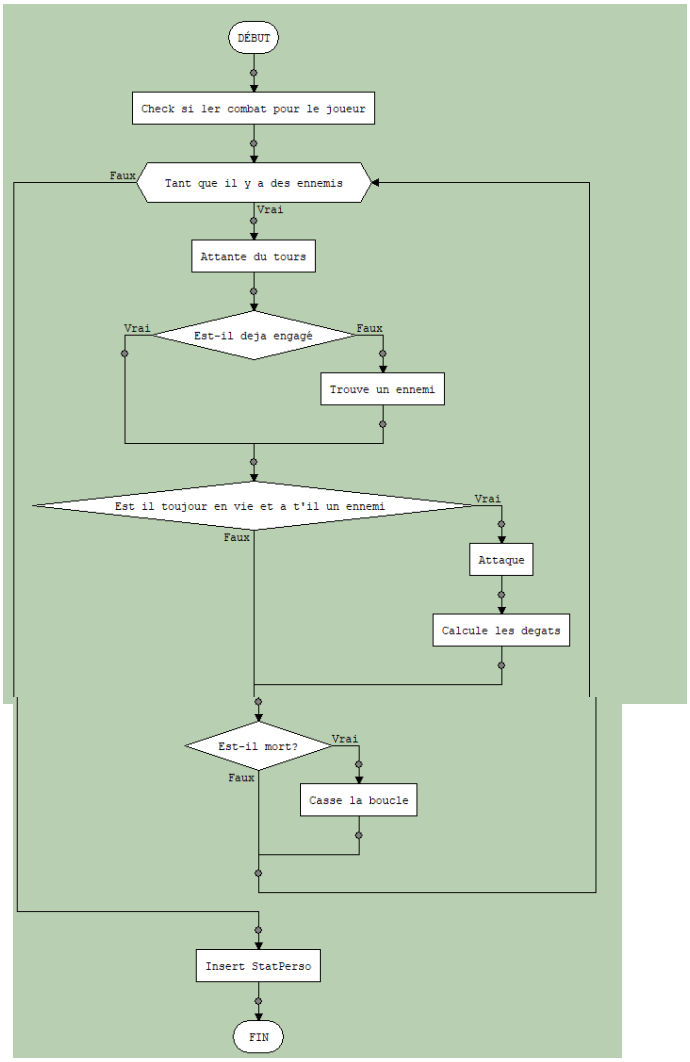


Algorithmes

Algorithme principal



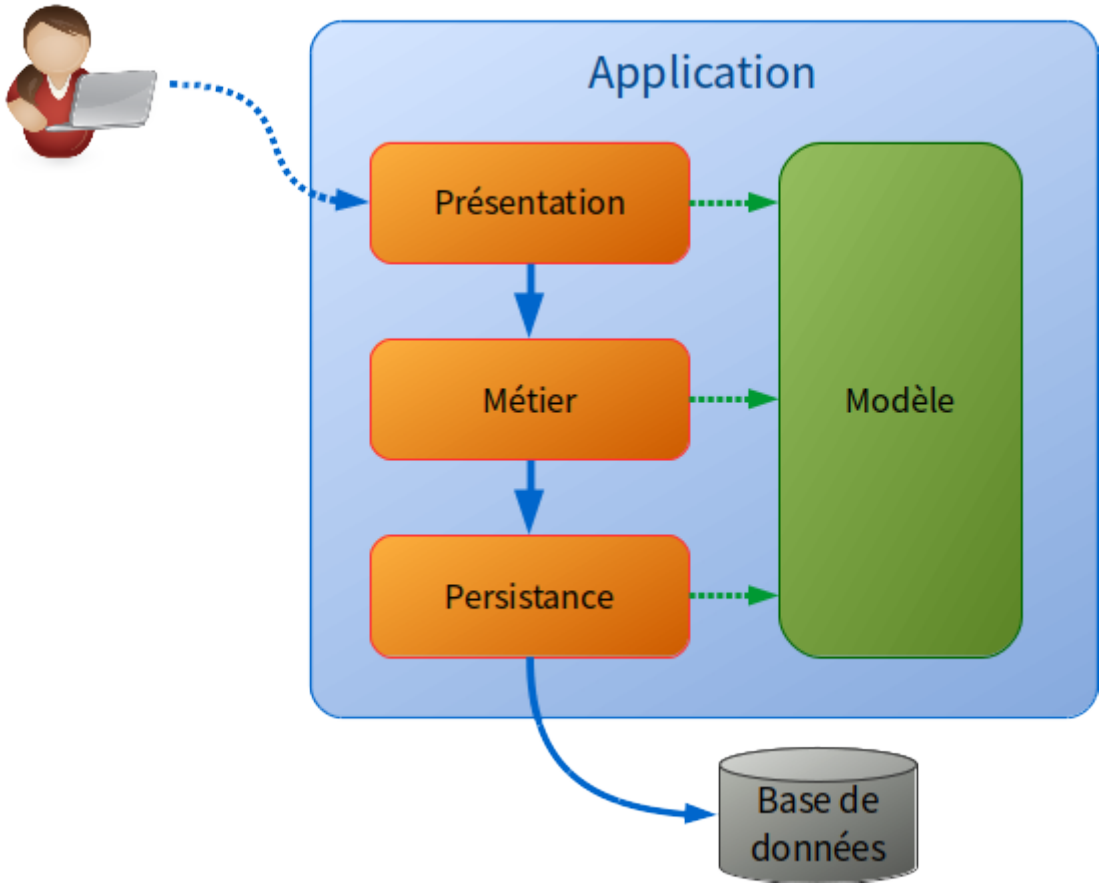
Algorithme de combat

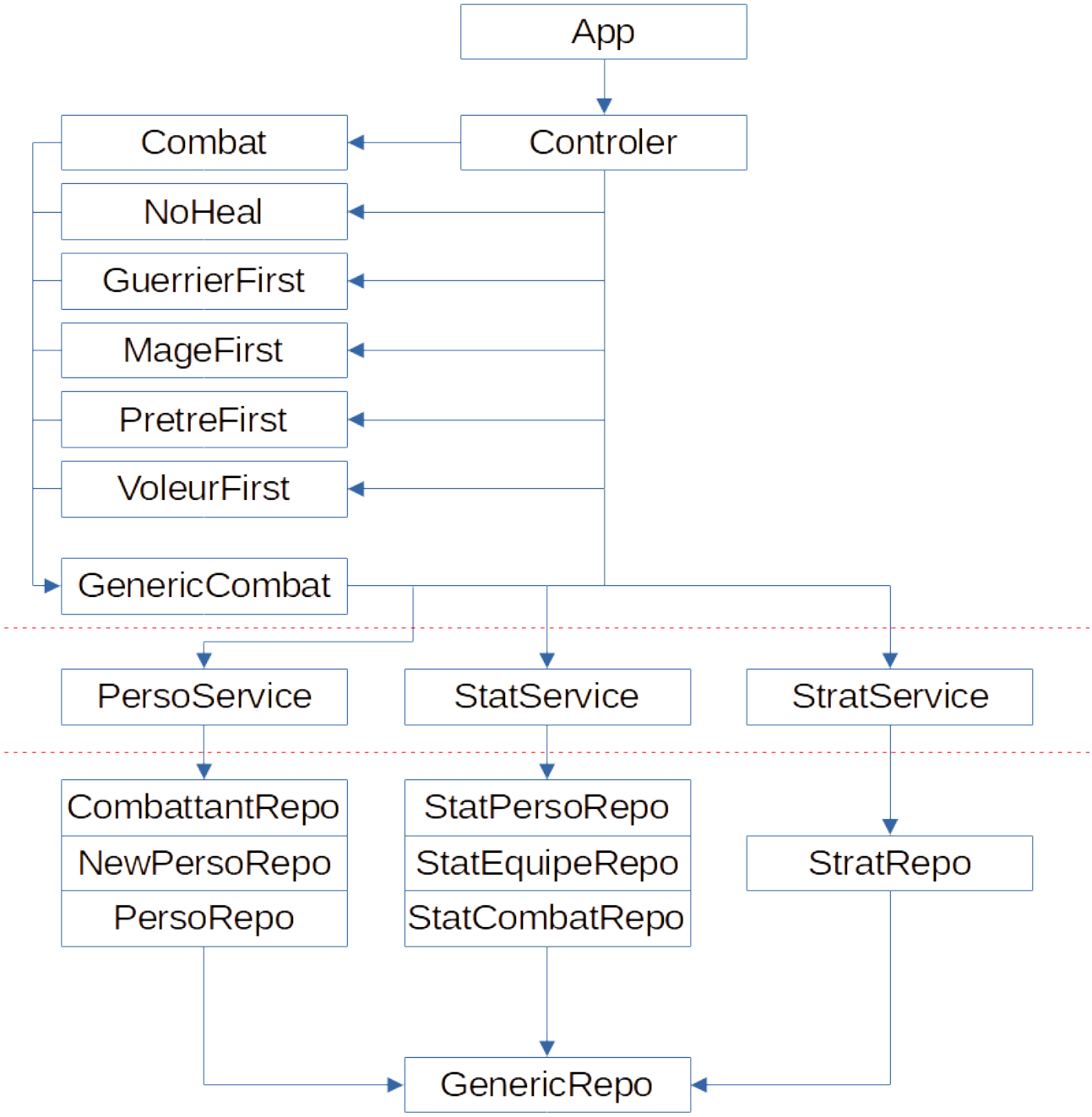


Architecture du projet

Le projet est développé sous le modèle d'architecture multi-tiers avec Maven. Elle fonctionne avec quatre couches. La couche application (`Rpg-App`). C'est la couche qui permet l'interaction avec l'utilisateur. La couche en dessous est la couche métier (`Rpg-Bll`). Elle est responsable de l'implémentation des règles de gestion fonctionnelles. La troisième couche, nomé couche persistence (`Rpg-Dal`) est la couche qui permet une communication avec la base de donné.

Toutes ces couches partagent une autre couche, la couche modèle (`Rpg-Model`). C'est dans cette couche que ce trouve les `Data-Object`.





En plus des ces couches, deux Pakages viennent s'ajouter au projet. Le premier (`Tools`) contient différentes méthodes générique qui peuvent s'implémenter à n'importe quel projet. Il a pour but de faciliter le développent. Le second Pakage (`Rpg-Database`) permet une connexion avec la base de données.

Couche Rpg-App

Dans la couche Rpg-app, il n'y a aucun calcule, ou changement des modèles. Si le programme doit être amener à changer une donnée d'un modèle (par exemple la mise à jour des point de vies), le contrôleur envoi les informations au service adéquat est c'est lui qui fait les calcules et set les nouvelles données.

Controler

Controler.java

Le contrôleur est la la classe qui communique avec les services et les classes de combats.

Classes de Combats

Il y a 6 classes de combats plus une classe générique. Chacune apporte une stratégie différente de combats. Elles communique avec les services, mais ne font en aucun cas le moindre calcul ou modification de données dans les modèles. Ces tâches reviennent au services.

GenericCombat.java

Cette classe est la classe générique. Elle contient les méthodes pour un combat basique. Toutes les autre classes de combat en hérite.

Combat.java

C'est la classe qui donne un combat standard. L'équipe attaque n'importe quelle ennemi. Elle ne contient pas de méthode précise car utilise toutes les méthodes de la classe générique.

NoHeal.java

Pareille que le classe du haut à la différence que les prêtre ne guérissent pas les alliés. La méthode `Attaque()` est différente et n'est donc pas celle présente dans la classe générique.

GuerrierFirst.java

L'équipe va ce concentrer sur l'attaque de guerriers avant de cibler les autres classes. Elle utilise donc une méthode `Agro()` à elle.

MageFirst.java

L'équipe va ce concentrer sur l'attaque de mages avant de cibler les autres classes. Elle utilise donc une méthode `Agro()` à elle.

PretreFirst.java

L'équipe va ce concentrer sur l'attaque de prêtres avant de cibler les autres classes. Elle utilise donc une méthode `Agro()` à elle.

VoleurFirst.java

L'équipe va ce concentrer sur l'attaque de voleurs avant de cibler les autres classes. Elle utilise donc une méthode `Agro()` à elle.

Couche Rpg-Bll

Services

PersoService.java

Il s'agit du service qui s'occupe de la création et de l'enregistrement des personnages avant et après chaque combats. C'est aussi ce services qui va modifier les données des modèles relatifs au personnages.

StatService.java

Il s'agit du service qui s'occupe d'enregistrer les statistiques. C'est aussi dans ce services que les modifications des modèles relatifs au statistiques sont modifiés.

StratServices.java

Ce service s'occupe d'aller chercher une stratégie aléatoire pour les équipes. Il n'y a pas besoin de modification de données pour les modèles de stratégie.

Couche Rpg-Dal

Repositries

Les repository s'occupe de communiquer avec la base de donnée. Chacun d'entre eux hérite du `GenericRepo.java` dans le quel contient toutes les méthodes sql.

Il y a un repository par table vue et procédure stocké utilisé dans le projet.

Nom de la table	*Nom du repository*	*Description*
perso	PersoRepo.java	Table qui contient tout les personnages créer.
combattant	CombattantRepo.java	Vue qui donne la liste de manière aléatoire des personnages disposé à ce batte.
newPerso	NewPersoRepo.java	Procédure stocké qui crée un nouveau personnage.
stat_combat	StatCombatRepo.java	Table qui contient les statistiques relatives aux combats.
stat_equipe	StatEquipeRepo.java	Table qui contient les statistiques relatives aux équipes.
stat_perso	StatPersoRepo.java	Tables qui contient les statistiques relatives aux personnages.
strat	StratRepo.java	Table qui contient les données des différentes stratégies développé.

GenericRepo.java

Il s'agit d'un repository dont chaque autre repo en hérite. Il contient les query de bases afin de ne pas devoir récrire les query SQL à chaque fois qu'on en a besoins.

Méthode	***Description***
`List<Object> GetByID(int)`	Récupère la ligne selon l'id
`List<List<Object>> GetAll()`	Récupère tout le tableau
`List<List<Object>> GetAllWhere(String)`	Récupère tout le tableau selon des conditions
`List<List<Object>> GetColumn(List<String>)`	Récupère seulement les colonnes souhaité
`List<List<Object>> GetColumn(List<String>, String)`	Récupère les colonnes souhaité selon des conditions
`List<Object> GetFirst (String)`	Récupère une seule ligne selon des conditions
`List<Object> GetFirst(List<String>, String)`	Récupère une ligne des colonnes souhaité selon des condition
`Object GetOneRandom()`	Récupère une ligne au hasard de la table.
`Object GetOneRandom(String)`	Récupère une ligne au hasard selon des conditions
`int Count()`	Compte le nombre de lignes dans le tableau
`int Count(String)`	Compte le nombre de lignes avec des conditions
`Object UseStorProc()`	Exécute une procédure stocké et retourne son résultat.
`int Update(Object)`	Met à jours les champ de toutes la table. Retourne une réussite ou non.
`int Update(Object, String)`	Met à jours les champ de la table qui rentre dans la condition.Retourne une réussite ou non.
`Object Insert(Object)`	Insert les données dans la table et retourne la clef primaire générée.

Couche Rpg-Model

La couche Rpg-Model contient les différant modèles utilisé pour le programme. Les modèles qui se trouvent dans le dossiers databaseModel, sont les modèles qui sont utiliser pour communiquer avec la base de données. DetailCombattant.java et Equipe.java sont deux modèles qui sont utilisé pour le programme.