Génie Logiciel et Gestion de Projet

Sprint #2 :

[M5]Gestion du renforcement de zone déjà capturée.

LP SIL IDSE

Groupe D : Team Trompette

ROBEZ Ludovic

MKENINI Ismael

CHENE Emmanuel

ADLAL Mohamed-Mehdi

[Dans ce rapport nous détaillerons ce qui était attendu de notre groupe pour le sprint 2, chaque étape et choix qui ont été mis en place pour la bonne réalisation de notre module, accompagné d'une prise de recul ainsi que de nombreuses illustrations Uml pour nous aider à la construction et au développement de la gestion du renforcement de zone.]

Sommaire

I. Organisation de l'équipe

La répartition des tâches a été faite en fonction des personnes qui se débrouillent le mieux cotés serveur ou client.

Ludovic : Passage de la création des Shapes par le service de la logique du jeu et gestion du renforcement de la zone.

Mohamed Mehdi : Afficher le niveau et l'équipe à laquelle la zone appartient en cliquant dessus.

Emmanuel : Empêcher le joueur de renforcer la zone autant de fois qu'il le souhaite.

Ismael : Afficher une notification de capture de zone aux membres de l'équipe et mise en place d'une BDD.

Utilisation de Jira Importante , Plusieurs branches dans git pour la partie développement

II. Explication des spécification

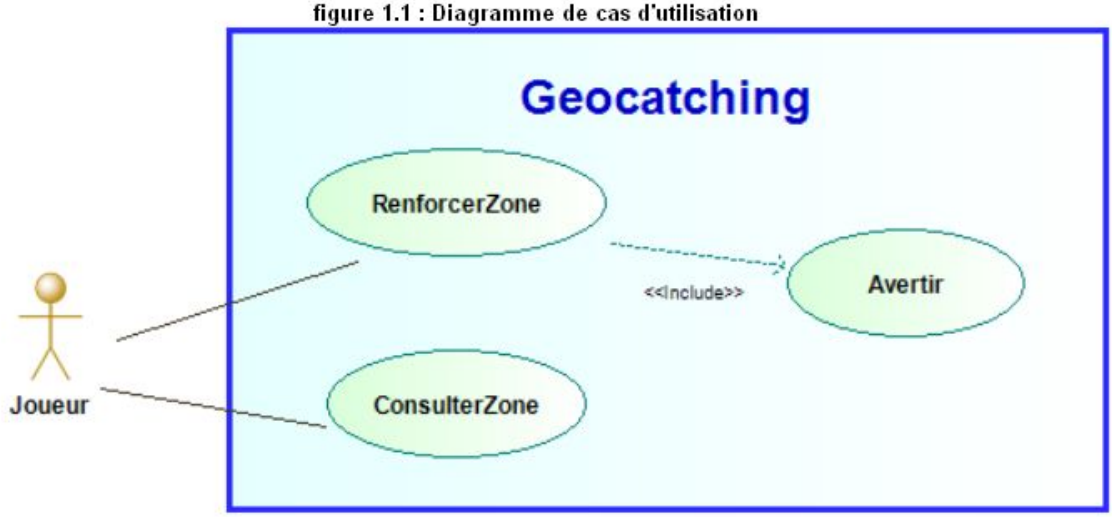
A. Présentation générale du module

Ce module est responsable du renforcement d'une zone. Le joueur doit posséder un certain nombre de point qui varie selon le niveau de la zone et ne pas avoir déjà renforcé la zone pour pouvoir renforcer une zone de son équipe. Le renforcement de la zone est représenté sous la forme d'un niveau. Selon le temps passé à chercher la photo du mini-jeu (voir Team E: The Hash Team), le renforcement peut être plus ou moins grand.

Le joueur doit être averti quand il réussit un renforcement. De plus, n'importe quel joueur doit pouvoir voir , à n'importe quel moment de la partie, les informations suivantes: le nom de la zone, son niveau après le renforcement ainsi que le nom de l'équipe à laquelle appartient.

Les services rendus par ce module sont représentés par ce diagramme de cas d'utilisation :

figure 1.1 : Diagramme de cas d'utilisation



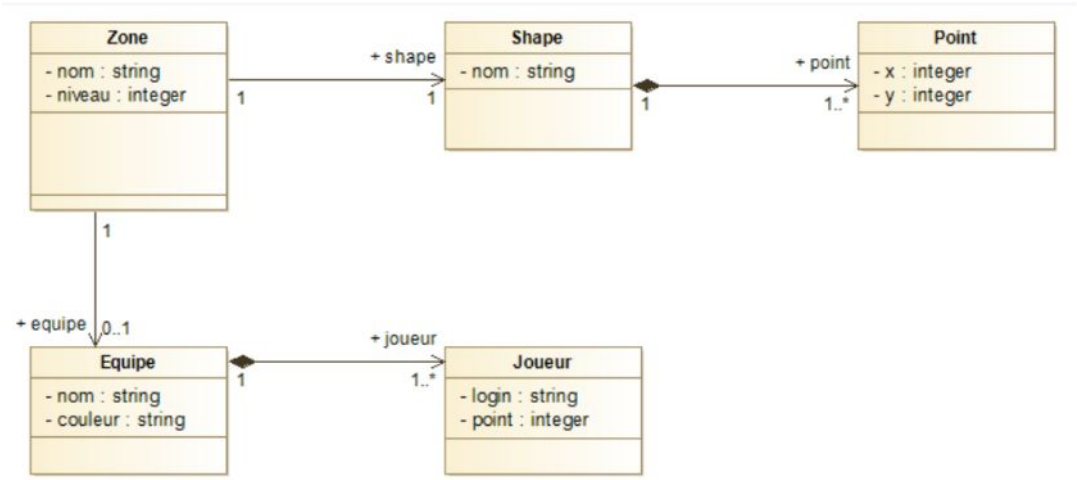
De plus nous avons réalisé un Diagramme de Classe du Domaine afin de synthétiser notre diagramme de cas d'utilisation (figure 1.1) :

- Diagramme de classe du domaine :

La phase d'analyse permet d'élaborer la première version du diagramme de classes, que l'on peut nommer diagramme de classe du domaine. Ce diagramme permet donc de modéliser les futures classes métiers de notre module .

Ces classes du domaine peuvent être identifié directement par les liens entretenue et comme on peut le voir sur le diagramme. Ce diagramme doit représenter d'une manière simple les spécification du cahier des charges.

figure 1.2 : Diagramme de classe du domaine

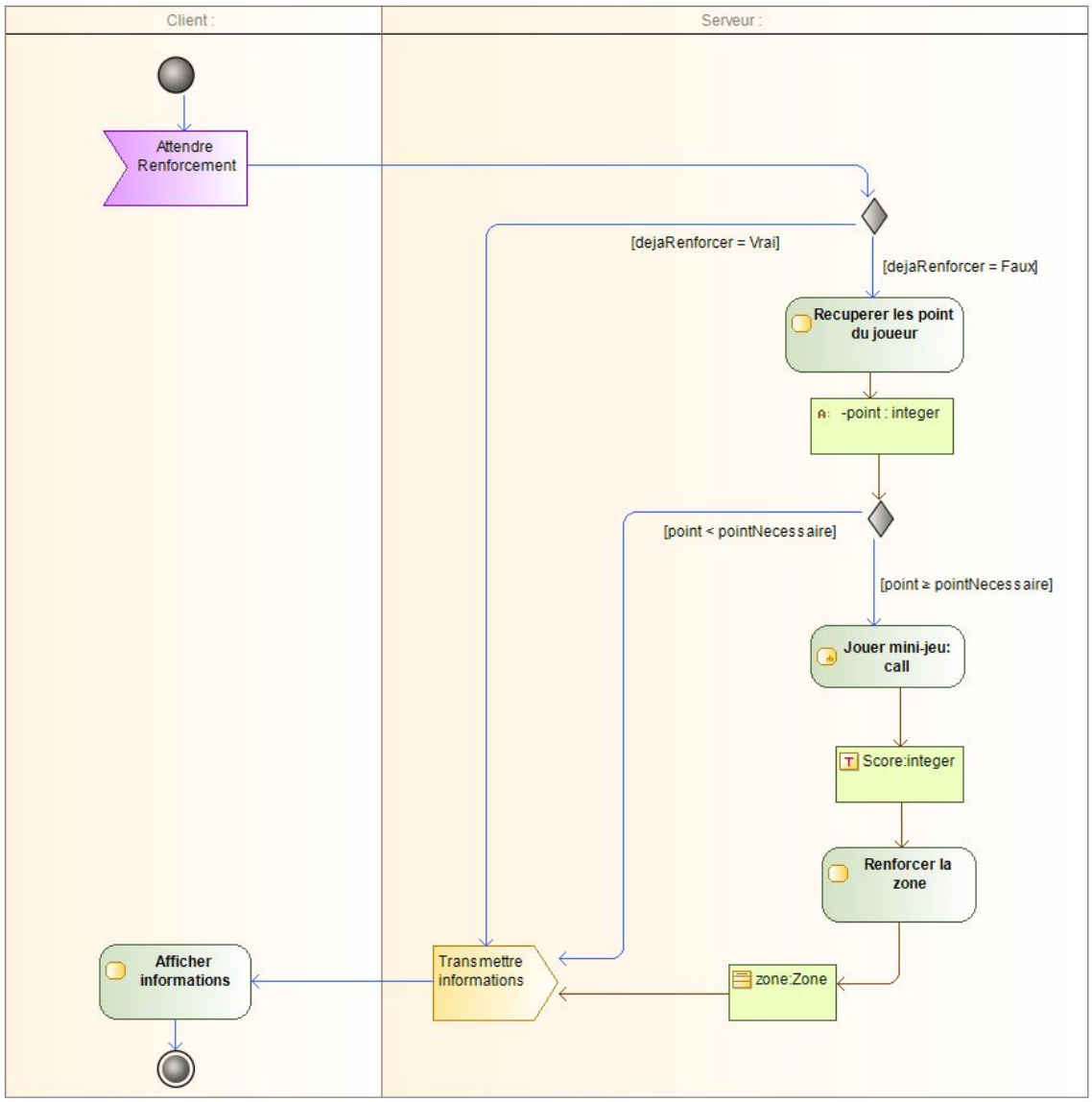


B. Etude sur le fonctionnement du module

Afin de synthétiser le fonctionnement du module, nous avons réalisé deux diagrammes d'activités permettant de représenter le déclenchement d'événements en fonction des états du système :

- Use Case: RenforcerZone

figure 1.3 : Diagramme d'activité du cas d'utilisation renforcer

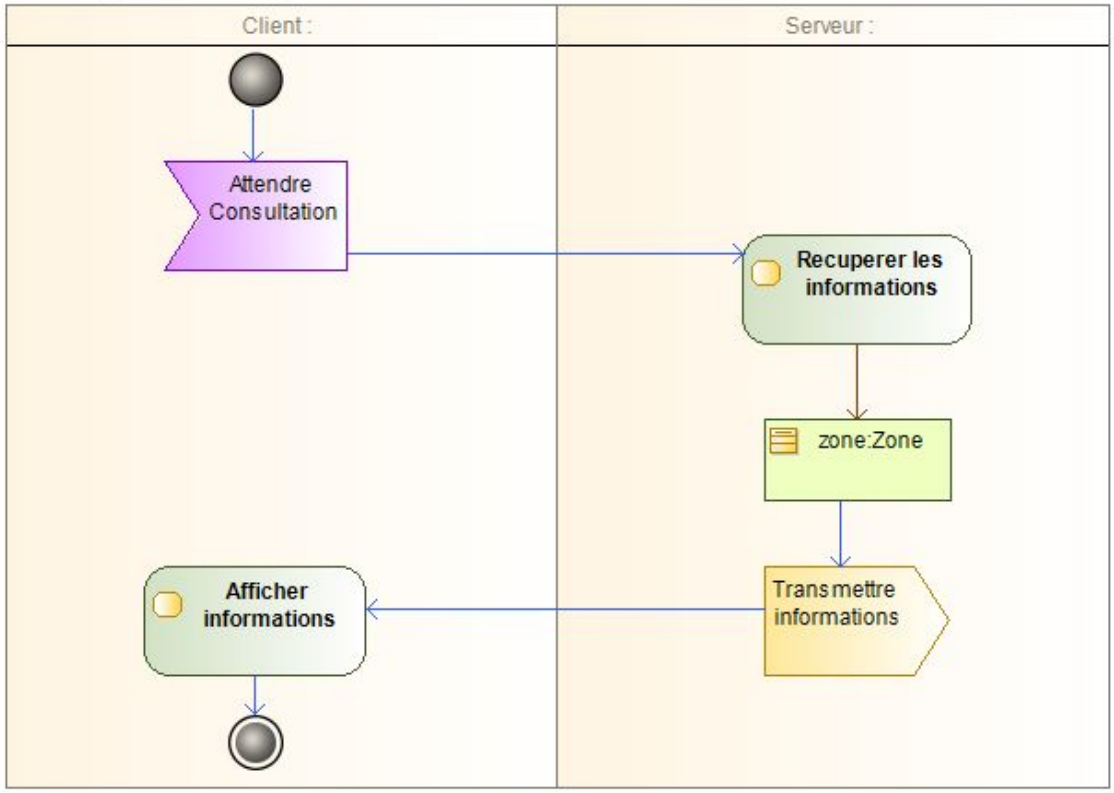


Lorsque l'événement “Renforcer” est appelé, nous allons vérifier que le joueur qui a appelé l'événement n’a pas déjà renforcer la zone auparavant. Ensuite nous vérifions si ce joueur possède assez de point pour pouvoir renforcer sa zone.

On appelle le Mini-jeu (voir Team E) selon le résultat de celui-ci nous augmentons le niveau de la zone en conséquence. Et on avertit l’utilisateur en cas de réussite ou d'échec.

- Use Case: ConsulterZone

figure 1.4 : Diagramme d'activité du cas d'utilisation ConsulterZone



-

Si un joueur souhaite avoir les informations sur une zone, nous récupérons le nom, le niveau et l'équipe a qui appartient la zone. Ensuite on envoie les information côté client qui s'occupera de les afficher.

C. Analyse détaillée du système

Nous avons séparé le jeu en 2 services:

- Shape Service :

Le service de forme, il regroupe toutes les classes utilisés pour la gestion des formes géométrique qui seront affiché et utilisé dans notre jeux.

- Game logic Service :

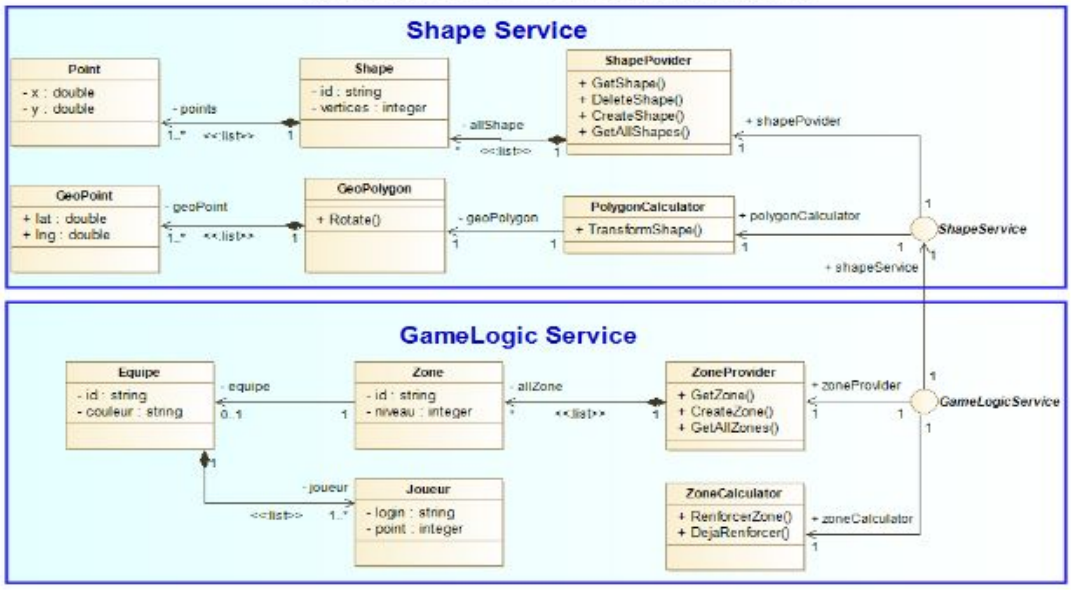
Ce package concerne tout les classes permettant de gérer la logiques du jeu lié à notre module. Le diagramme de classe d'analyse sert à modéliser plus en détails le fonctionnement de nos classes métiers et de visualiser les classes qui serviront de classe d'extension à nos classes métiers (voir classes Providers).

Ainsi grâce à ce diagramme (figure 1.5) il est possible d'identifier les concepts et d'organiser et simplifier notre futur modèle final qui sera un diagramme de classe de conception (rétro ingénierie du code). De plus, nos classes sont rangés en paquetage afin de les rassembler selon leur tâches communes.

Les packages permettent plus de clarté et de visualiser où se situe chaques classes dans la conception du module.

Le Service Shape introduire la notion de zone , La gestion des notifications permettant d'avertir le joueur d'un renforcement de zone effectué dans la partie ainsi que les restriction au re-renforcement de zone.

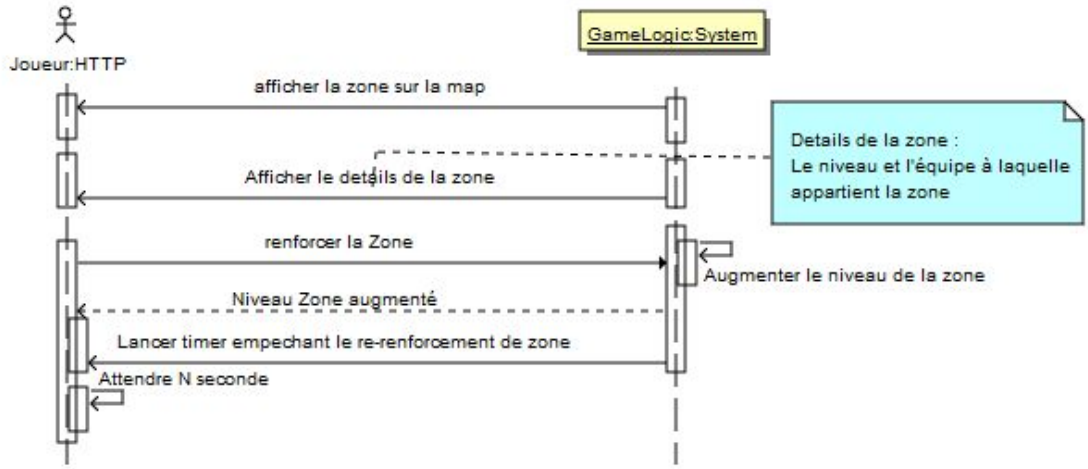
figure 1.5 Diagramme de classe d'analyse



-Diagramme de Séquence :

Le diagramme de séquence système permet de décrire le comportement et les interactions entre le système, qui est ici le service gameLogic , et les acteurs représenté ici par le joueur.

figure 2.1 : Diagramme de Sequence Systeme



Description :

Comme nous le montre (figure 2.1) , le système récupère les formes et les affiches sur la map ou l'acteur se trouve. Il est important de noter tout de même que la zone est déjà en état de capture par l'équipe dont le joueur fait partie.

Une fois la forme bien déployé sur la map , le joueur peut renforcer la zone en y entrant dedans et déclencher "l'événement renforcer la zone". Le système GameLogic va traiter les informations concernant le renforcement de zone en augmentant le niveau.

On peut voir (figure 2.1) alors que le joueur doit alors attendre un certain temps pour pouvoir répéter l'action de renforcement de zone . Le joueur peut également voir à tout moment les détails concernant la zone , comprenant : le nom de la zone et son niveau ainsi que l'équipe qui l'a capturé.

-Diagramme de séquence système :

Le diagramme de séquence pour la conception est utilisé pour donner une vue en largeur du fonctionnement du système des appels de méthodes et de la coopération entre instances et méthodes . Comme on peut voire sur le diagramme (figure 2.2) on remarque que par rapport au précédent diagramme de séquence (figure 2.1) , on a remplacé le système par des objets et on donne plus de détails sur les méthodes concernés. Ce diagramme se rapprocher plus du code et des algorithmes mis en place pour les solutions mis en place dans la conception de notre module.

Description : (figure 2.2)

On peut voir donc tout d'abord que les principaux messages échangés entre client et serveur sont effectué entre la partie Joueur(client) et GameLogic(Serveur). Nous avons opté pour plusieurs classes se servant du service Web RESTful :

GameLogic:HTTP (figure 2.2) fait l'intermédiaire entre gameLogic:Zone et gameLogic:Notif et s'occupe de la communication HTTP avec la partie client.

Gamelogic:Zone et gameLogic:Notif (figure 2.2) effectue les actions nécessaire en amont grâce à leur méthodes ,

Chaque requête http effectué est retourné par un code de retour , 200 OK dans le cas nominal(succès de la requête) et 400 (requête incorrect) pour le scénario d'exception modélisé ici par le fragment break.

Lorsque le client accède à la map, les zones sont récupérées côté Web Service par le gameLogic de zone , qui passe également par le service de forme "shapeService" afin de produire les formes géométriques qui seront assimilées en tant que Zone dans notre jeux.

Lors du renforcement de zone , le joueur envoie une requête HTTP afin que le cotés serveur mette à jour le niveau de la zone dans la BDD dans la table “zone” grâce aux requêtes SQL (UPDATE, SELECT ...).

GameLogic:Notif (figure 2.2) , se comportant comme un observateur de la zone ,intervient dans le système afin de récupérer les informations du renforcement de zone en communiquant avec gameLogic:Zone , et déclanche la méthode Notif() côtés client qui permettra au joueur d'avoir une notification pour lui confirmer que la zone a bien été renforcée.

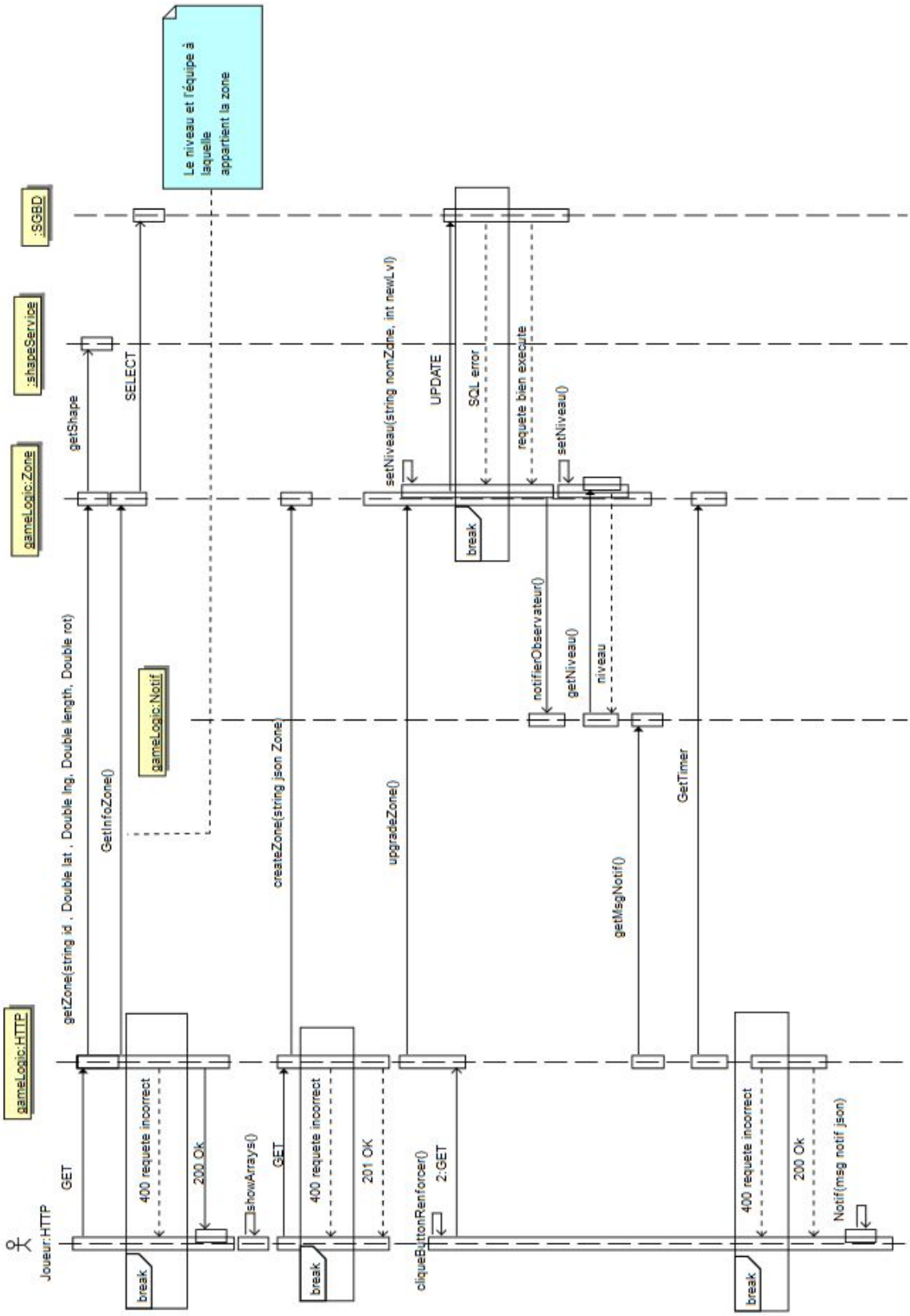


figure 2.2 : Diagramme de Sequence de Conception

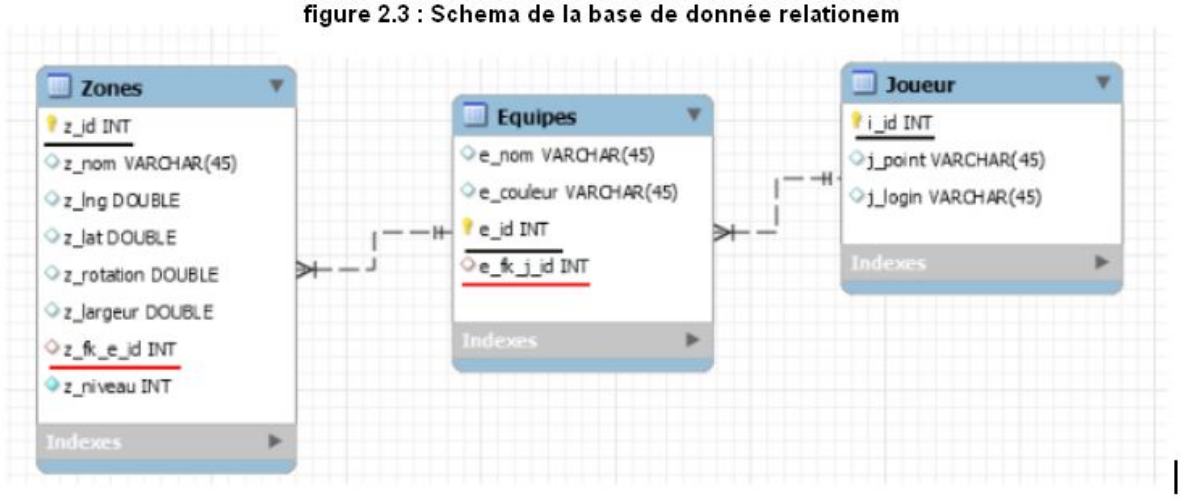
III. Justification des choix conceptuels

A. Persistance des données

Pour la persistance des données, nous avons choisis de créer une base de données avec pour Système de Gestion de Base de Données MySQL

Notre base de donnée est assez simple, comprend trois tables : la tables **Zones** et **Equipes** et **Joueur.** Ces tables stockent les informations nécessaires au bon développement du module.

figure 2.3 : Schema de la base de donnée relationem



La figure 2.3 représente donc la relation entre les tables. On peut voir les clés primaires soulignés en noire (z_id , e_id et j_id) la clé étrangère en rouge (z_fk_e_id et e_fk_j_id).

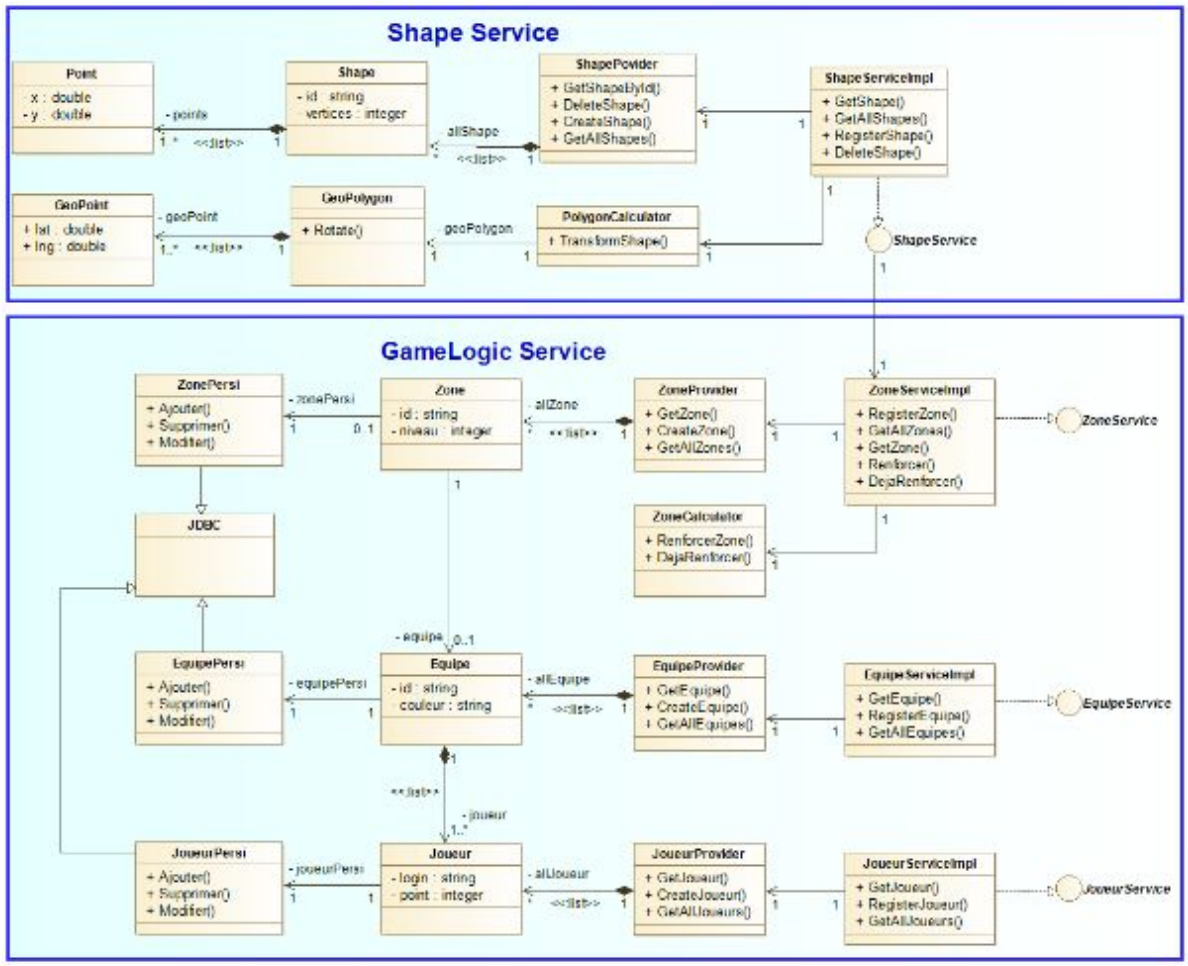
-La table zone contient une clé étrangère équipe afin d'exprimer la capture de zone d'une équipe , elle est nulle si la zone n'est pas capturé .

-Lors de l'exécution du renforcement de zone, une requête est alors effectuée vers la table Zones où le niveau de la zone augmente en fonction de son nom,

Nous utilisons d'ailleurs des classes Persi (figure 1.5) permettant d'effectuer les requêtes Sql et de stocker les résultats de certaines (SELECT, DELETE, UPDATE, INSERT) .

B. Architecture côté Serveur

Afin de mieux vous montrez notre architecture coté serveur, nous avons réalisé un Diagramme de classe montrant les différents services.



Nous avons fait ce choix afin de bien diviser les différentes classes métiers. Chaque classes métiers possèdent sa propre interface afin de pouvoir récupérer n'importe quel chose à n'importe quel moment du côté client. Cela nous offre un côté client plus facile a réalisé et donc un gain de temps énorme autant côté Client que pour la programmation de fonction lié au renforcement de zone.

C. Les avantages et inconvénients

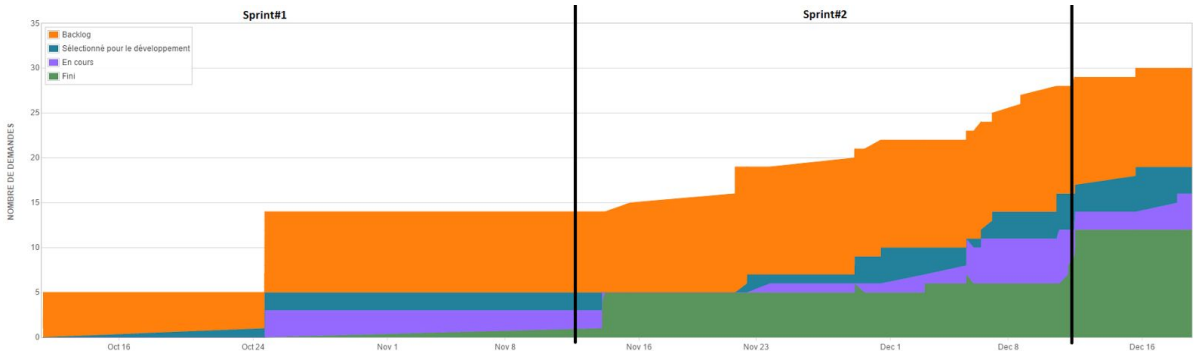
Comme nous l'avons cité au dessus notre architecture côté serveur, permet une flexibilité côté Client qui n'est pas négligeable. Cela nous permet d'utiliser n'importe quel classe métier quand cela nous en avons besoin côté Client. De plus, La persistance des données est un avantage car la partie est sauvegardée cela permet à l'utilisateur de revenir jouer a n'importe quel moment sans devoir recommencer à zéro.

IV. Rétrospective entre le Sprint#2 et Sprint#1

Le sprint 2 aura été différent en terme d'organisation. En effet nous avons réussi à mieux gérer la gestion des tickets sur Jira, nous avons également développé une meilleure communication qu'au sprint 1. Au niveau des fonctionnalités nous avons fait des tests de capture et renforcement de zone avec déplacement de deux marker pour simuler deux joueurs entrant dans une zone. Egalement nous avons corrigé le problème que nous avions avec Jenkins.

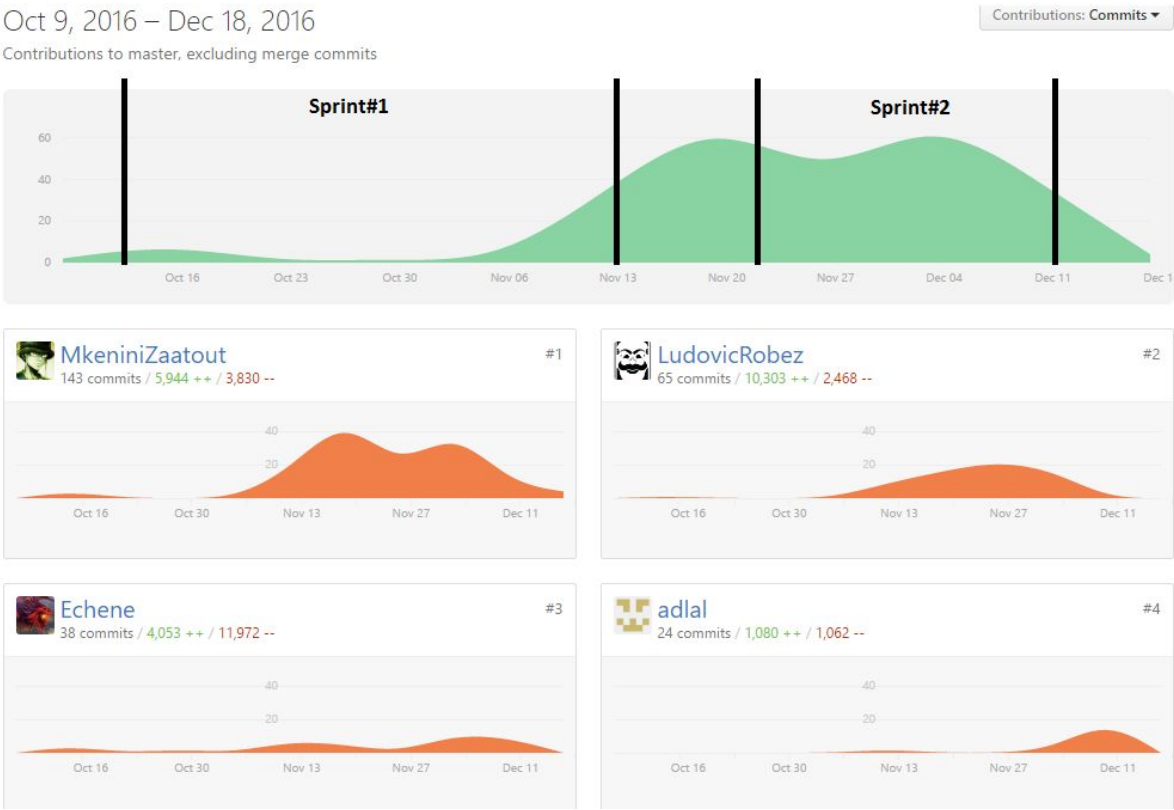
Suite à l'architecture serveur que nous avons choisi, nous avons dû développer différents services. Les services de joueur d'équipe et de zone. Ce service de zone utilise le service de shapes "ws-shapes" car ils fonctionnent ensemble.

Au niveau des outils de gestion logiciel nous avons plus utilisé Jira au niveau du ticketing, les commits sur Git y sont associé avec le code sur la tâche Jira, cela permet d'avoir un traçage plus lisible des avancées du projet et de savoir où tout le monde en est. La description des commits est également plus claire qu'au sprint 1.



Ici nous pouvons voir que le nombre de commits est beaucoup plus conséquent au sprint 2 par rapport au sprint 1, nous avons eu une meilleure organisation à ce niveau là et cela se reflète sur ce graphique.

Team Trompette



De plus, le nombre de test entre le sprint#1 et le sprint#2, nous sommes passés de 2 test fonctionnel à 11 test.

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running shape.GeoPolygoneTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.056 sec
Running shape.ShapesProvidersTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec

Results :

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running joueur.JoueursProviderTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.055 sec
Running zone.ZonesProviderTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 sec
Running zone.ZonesCalculatorTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.009 sec
Running equipe.EquipesProviderTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec

Results :

Tests run: 8, Failures: 0, Errors: 0, Skipped: 0
```

V. Perspectives

A. Améliorations

1) General

Depuis les premiers sprint nous avons amélioré notre communication au sein du groupe. En effet la communication au sein des premiers sprint était déplorable. Nous travaillons chacun de notre côté en 2 groupes créés au départ pour le tout premier sprint essentiellement. Nous avions des difficulté à se voir en dehors des cours pour travailler ensemble dans une bonne cohésion. Concernant les outils de gestion de projet logiciel nous avons mieux géré la gestion de tickets avec Jira.

Pour le projet en lui même il y'a des choses à améliorer. Le renforcement de zone fonctionne et on ne peut renforcer la zone qu'une seule fois. La fonctionnalité pour le click sur une zone n'est pas encore fonctionnelle nous projetons de la terminer d'ici peu. Nous projetons également de se réunir plus souvent pour palier aux différents soucis organisationnels engendrés par le non rassemblement de notre groupe.

2) Développement

Nous allons essayer d'utiliser les different Design Pattern vu en cours dans notre module, par exemple:

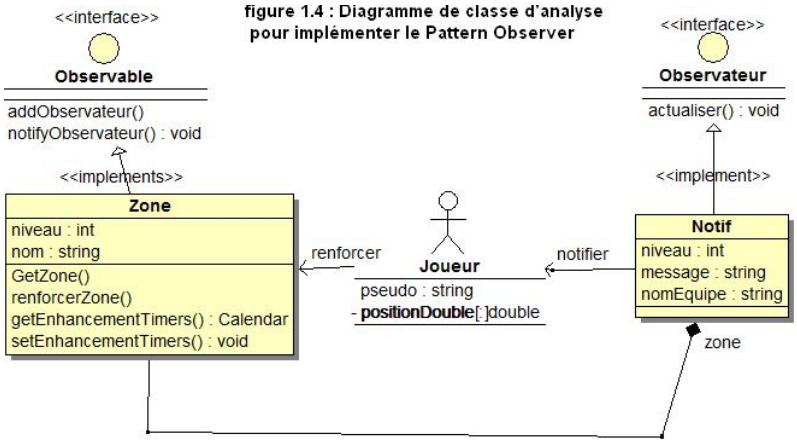
- Design Pattern: Singleton

Ce design pattern permettra de se connecter une seul fois au début a la Base de données afin d'éviter de le faire a chaque requête SQL.

Cela nous permettra de gagner en rapidité d'exécution en côté Serveur car la connexion a la base de données peut prendre plus ou moins de temps surtout si celle-ci est répété.

- Design Pattern: Observer

figure 1.4 : Diagramme de classe d'analyse pour implémenter le Pattern Observer



Nous avons également essayé d'implémenter le patron de conception Observer .Il définit une relation entre objets de façon que, si un objet change d'état, tous ceux qui en dépendent en soient informés et mis à jour automatiquement, Or c'est bien le cas dans notre module puisque l'état d'une zone change régulièrement nous souhaitons être notifié.

On a un donc (figure 1.4):

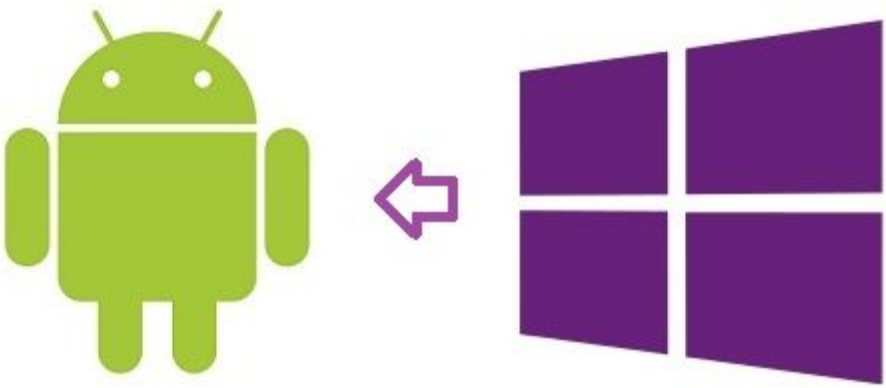
-l'Observable : La classe qui notifie les observateurs quand il change d'état (Zone).

-Les Observateurs : La classe qui acceptent les notifications (Notif).

les méthodes des interfaces Observable et Observateur permettront d'avertir tous les observateurs lors d'un changement d'état , d'ajouter un observateur ou d'actualiser l'état.

Le but serait de le finir et l'intégrer de notre module.

B. Que faire pour progresser vers la solution mobile



Il faut réussir à terminer le côté serveur afin qu'il soit opérationnel pour ne devoir refaire que la " façade " sur mobile. Il faut que nous ayons le moins de fonctionnalité et d'intelligence côté client et tout faire côté serveur. Un gain de temps énorme sera effectué en procédant de cette manière et nous sommes sur la bonne voie.

D'un point de vue extérieur à notre situation actuelle dans ce projet , un portage du projet sur une nouvelle plate-forme qu'est la solution mobile nécessite de la repenser et de la redévelopper dans les grandes lignes car il s'agit de langages et d'interfaces radicalement différents.

Il faudra également analyser et cerner nos besoins, prôner des pistes ergonomiques en fonction de la plate-forme.

VI. Réflexions et conclusions

Pour ce projet nous avons travaillé en équipe c'est à dire que le projet à été divisé en 4 parties. Les formes lié au service de forme sont désormais intégrées à la logique du jeu, les requêtes SQL pour enregistrer les informations concernant le renforcement dans la BDD de la zone s'exécute, nous sommes désormais plus à l'aise avec un projet maven , nous utilisons des classes de test constamment pour nous aider à définir nos futur problème lié à la conception de notre code.

Cependant notre module dépend de plusieurs autres modules de groupe il nous reste à intégrer notre l'intégrer aux autres groupes.

Dans l'ensemble cela fut instructif de travailler en groupe, l'entraide et l'écoute de ses camarades est la base dans un projet comme celui-ci.