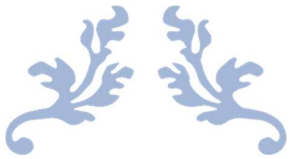


SUPER PONG

Mini-projet de GEN

Créé par:

Jérémie Chatillon
Antoine Rochat
Benoît Schopfer
James Smith

15 JUIN 2018

VERSION 1.1

Table des matières

Introduction

Le but de notre projet est de développer un jeu similaire au Pong[1] permettant de jouer à des parties multijoueur en ligne ou de jouer en local contre une IA, ceci avec deux modes de jeu à choix : le mode classique et le mode avec des objets (items). Le serveur ainsi que le client ont été testé sur *Windows*.

Fonctionnement général :

Utilisateur :

Un utilisateur pourra choisir entre jouer en local contre une intelligence artificielle ou se connecter et jouer en ligne. En ligne, le joueur pourra rejoindre des parties allant de 2 à 8 joueurs. Un joueur pourra choisir le nombre d'adversaires souhaité ainsi que s'il veut jouer avec des objets ou non. Les objets apparaitront de façon aléatoire sur le terrain et seront activés lorsque la balle les atteindra. Ils auront divers effets momentanés tels qu'accélérer la balle, ralentir la raquette de l'adversaire etc.

Administrateur :

Un administrateur pourra modifier les statistiques des joueurs ainsi que réinitialiser leur mot de passe. Il aura également toutes les possibilités des joueurs normaux.

Base de données :

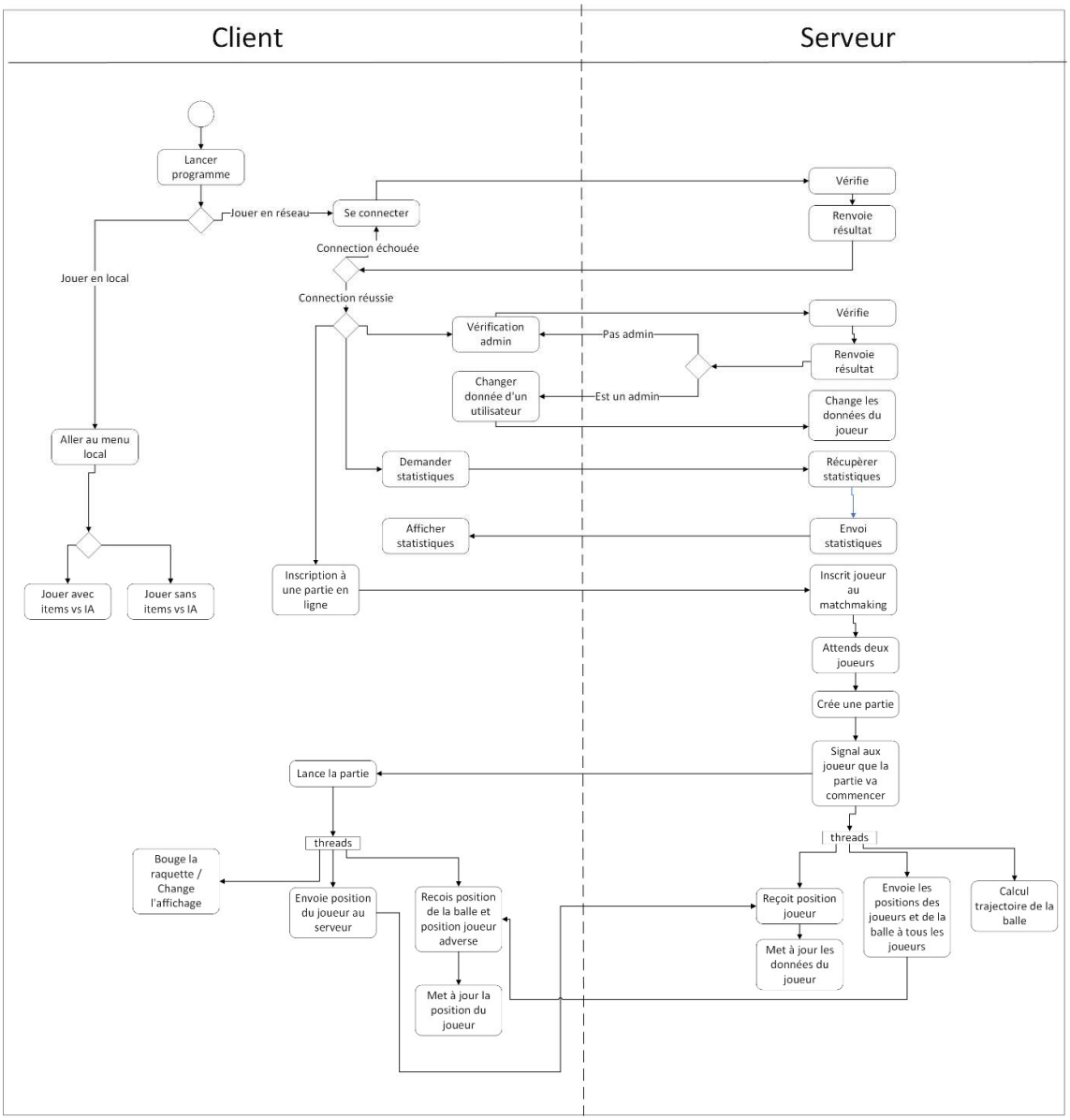
La base de données contiendra la liste des utilisateurs inscrits, leur mot de passe, leurs droits (utilisateur ou administrateur) ainsi que leurs statistiques.

Mockups de l'interface :

Pour les différents mockups de l'interface, veuillez-vous référer au manuel d'utilisation fourni en annexe qui présentent les différents menus de l'application ainsi que les possibilités qui sont offertes à l'utilisateur.

[1] https://en.wikipedia.org/wiki/Pong

Diagramme d'activité

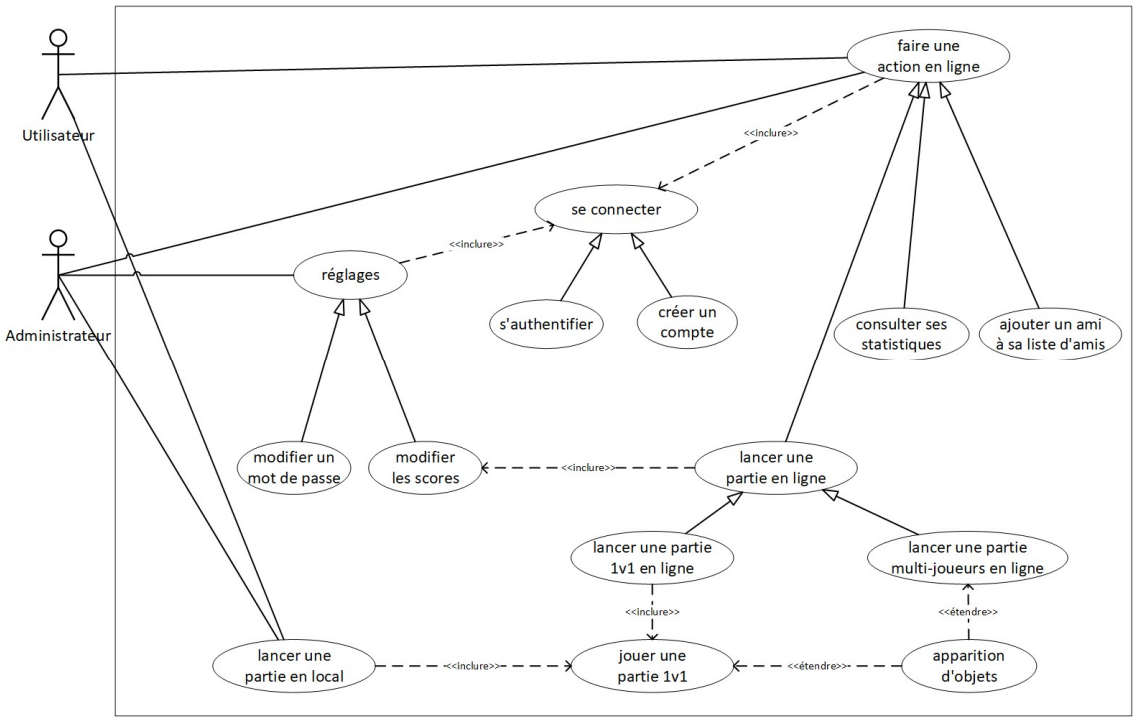


Nous avons fait un diagramme d'activité simplifié qui illustre le partage des responsabilités entre le serveur et le client. On a les interactions suivantes entre le client et le serveur :

- Lorsque le client veut se connecter en ligne, il envoie son nom d'utilisateur ainsi que son mot de passe au serveur, ce dernier lui répond si la connexion a réussie ou échouée.
- Lorsque le client veut visualiser ses statistiques, il envoie la demande au serveur qui lui renvoie ses statistiques.
- Lorsque le joueur veut jouer une partie de Pong en ligne, il envoie une demande au serveur qui lui va inscrire le joueur à un *matchmaking* jusqu'à ce qu'il y ait assez de joueurs qui ont fait une demande de partie. Dès lors, le serveur crée une nouvelle partie et signale au client qu'une nouvelle partie de Pong va débuter.

- Lorsqu'une nouvelle partie est lancée, le joueur envoie périodiquement la position de sa raquette au serveur qui met à jour la position du joueur de son côté. Le serveur lui envoie périodiquement la position des joueurs et de la balle au client, qui lui met à jour la position du joueur adverse ainsi que la position de la balle.

Diagramme des cas d'utilisation :



Cas d'utilisation :

Lancer le programme :

Prérequis : L'utilisateur lance le jeu de Pong.

1. Il doit choisir en 2 options :
   a. Lancer une partie en local -> suivre scénario *Lancer une partie en local* :.
   b. Faire une action en ligne -> suivre scénario *Faire une action en ligne*.

Lancer une partie en local :

1. En local, seul le mode *1v1* (avec ou sans objets) est disponible.
2. Le joueur affronte une IA.
3. Suivre le scénario *Jouer une partie 1v1* :.
4. Retourner au point 2 du scénario *Lancer le programme* :.

Jouer une partie 1v1 :

1. Deux joueurs s'affrontent dans une partie de Pong.
   a. Un des deux joueurs peut être une intelligence artificielle.
2. Si l'option *avec objets* a été choisie -> Suivre le scénario *Apparition d'objets* :.
3. Un joueur gagne un point lorsque la balle dépasse la raquette de l'adversaire et atteint son mur.
4. Le premier joueur à 5 points a gagné.
5. Retourner au scénario appelant (*Lancer une partie en local* : ou *Lancer une partie 1v1 en ligne* :) et le continuer.

Apparition d'objets :

Prérequis : l'option *avec objets* a été choisie.

Tout au long de la partie, des objets apparaissent aléatoirement sur le terrain.

1. Lorsque la balle touche un de ses objets, un effet se déclenche pendant un temps limité et l'objet disparait du terrain.
    a. Exemple non exhaustif d'effets :
        i. Accélération de la balle.
        ii. Duplication de la balle pendant un temps défini.
        iii. Ralentissement du déplacement de la raquette de l'adversaire pendant un temps défini.
2. Si l'objet n'est pas atteint après un certain temps, il disparait.
3. Si un objet est atteint par la balle alors qu'un autre est en cours d'effet, les deux effets s'additionnent.
4. L'effet d'un objet disparait après quelques secondes.
5. Retourner au scénario appelant (*Jouer une partie 1v1* : ou *Lancer une partie multi-joueurs en ligne* :).

Faire une action en ligne :

1. Le joueur doit s'authentifier -> suivre le scénario *Se connecter* :
2. Une fois authentifié, il doit choisir entre 3 options
    a. Jouer en ligne -> suivre scénario *Lancer une partie en ligne* :
    b. Ajouter un ami à sa liste d'amis -> suivre scénario *Ajouter un ami à sa liste d'amis* :
    c. Consulter ses statistiques -> suivre scénario *Consulter ses statistiques* :
3. Si le joueur possède les droits d'administrateur, l'option supplémentaire *Réglages* est disponible. Si le joueur clique dessus, suivre le scénario *Réglages* :.

Se connecter :

1. Le joueur doit choisir entre deux options :
    a. S'authentifier (s'il possède déjà un login) -> Suivre le scénario *S'authentifier* :
    b. Créer un compte (s'il souhaite créer un nouveau compte) -> Suivre le scénario *Créer un compte* :
2. En fonction de son login, il se voit attribuer les droits d'administrateur ou d'utilisateur.
3. Retourner au point 2 du scénario *Faire une action en ligne*.

S'authentifier :

1. Pour s'authentifier, l'utilisateur doit entrer son login et son mot de passe.
    a. En cas d'erreur de connexion -> suivre le scénario d'erreur *Erreur de* connexion (scénario Se connecter :) :
    b. En cas d'erreur de login (login ou mot de passe invalide) -> suivre le scénario d'erreur *Erreur d'authentification* (scénario Se connecter :) :
2. Retourner au point 2 du scénario *Se connecter* :.

Créer un compte :

1. Le joueur entre ses informations de login, dont au moins un nom d'utilisateur et un mot de passe.
2. Le serveur vérifie que le nom d'utilisateur ne soit pas déjà utilisé (déjà présent dans sa base de données).
    a. Si le nom d'utilisateur est déjà utilisé, le joueur doit en entrer un nouveau.
    b. Si le nom d'utilisateur est disponible, le compte est enregistré avec les informations fournies par le joueur.
3. Si le compte a bien été créé, le joueur est automatiquement connecté avec ce login.

4. Retourner au point 2 du scénario *Se connecter* :.

Lancer une partie en ligne :

1. Le joueur a le choix entre deux types de partie :
    a. Partie *1v1* -> voir scénario *Lancer une partie 1v1 en ligne* :
    b. Partie *multi-joueurs* -> voir scénario *Lancer une partie multi-joueurs en ligne* :
2. À la fin de la partie, le(s) joueur(s) est(sont) renvoyé(s) au point 2 du scénario *Faire une action en ligne*.

Lancer une partie 1v1 en ligne :

Prérequis : Le joueur est authentifié et a choisi l'option *1v1* du point 2a du scénario *Lancer une partie en ligne* :.

1. Le joueur doit choisir entre 3 options :
    a. Défier un membre de sa liste d'amis.
    b. Jouer contre une IA.
    c. Jouer contre un joueur choisit par le serveur parmi les membres actuellement connectés en attente de jouer.
2. Dans les 3 cas, une partie *1v1* est créée par le serveur et les 2 joueurs s'affrontent.
3. Suivre le scénario *Jouer une partie 1v1* :.
4. Retourner au point 1 du scénario *Lancer une partie en ligne* :.

Lancer une partie multi-joueurs en ligne :

Prérequis : Au moins 3 joueurs s'affrontent dans une partie de Pong en ligne.

1. Le terrain est construit en fonction du nombre de joueur. Il y a autant de bord qu'il y a de joueurs.
    a. 3 joueurs -> le terrain est un triangle
    b. 4 joueurs -> le terrain est un carré
    c. 5 joueurs -> le terrain est un pentagone
    d. 6 joueurs -> le terrain est un hexagone
    e. 8 joueurs -> le terrain est un octogone
    f. Etc…
2. La partie commence.
3. La balle est mise en jeu au centre du terrain. Elle part dans une direction aléatoire, avec une vitesse constante fixée.
4. Si l'option *avec objets* a été choisie -> Suivre le scénario *Apparition d'objets* :.
5. Lorsque la balle dépasse la raquette d'un joueur et atteint son mur, il est éliminé mais il peut continuer à regarder la partie en tant que spectateur.
6. La balle s'arrête.
7. Le terrain est réduit en supprimant le mur du joueur éliminé. La nouvelle forme du terrain est déterminée en fonction du nombre de joueurs restants tel qu'expliqué au point 2.
8. Une fois le terrain recréé, le joueur ayant éliminé le dernier joueur supprimé engage la balle.
9. La partie continue en bouclant du point 4 au 9 jusqu'à ce qu'il n'y ait plus que 2 joueurs.
    a. À tout moment, si un joueur se déconnecte -> suivre le scénario d'erreur *Un joueur se déconnecte (scénario Lancer une partie en ligne :*)
10. Le terrain qui est alors construit correspond à un carré (comme pour les parties *1v1*).
11. Les 2 joueurs s'affrontent jusqu'à ce qu'il n'en reste plus qu'un.
12. Le dernier joueur restant est déclaré vainqueur.
13. Revenir au point 2 du scénario *Lancer une partie en ligne* :.

Ajouter un ami à sa liste d'amis :

Prérequis : Le joueur est authentifié et a choisi l'option *Ajouter un ami à sa liste d'amis* du point 2b du scénario *Faire une action en ligne.*

1. Le joueur peut entrer un pseudo.
2. Le serveur recherche ce pseudo parmi tous les joueurs inscrits.
    a. Si le pseudo correspond à un joueur inscrit, le serveur lie les 2 joueurs en les ajoutant réciproquement dans leur liste d'amis. Le joueur peut effectuer une nouvelle recherche.
    b. Si le pseudo n'existe pas, le serveur affiche une erreur et le joueur peut effectuer une nouvelle recherche.
3. Si le joueur appuie sur le bouton de retour, retourner au point 2 du scénario *Faire une action en ligne*.

Consulter ses statistiques :

Prérequis : Le joueur est authentifié et a choisi l'option *Consulter ses statistiques* du point 2c du scénario *Faire une action en ligne.*

a. Les statistiques (nombre de partie jouées, nombre de victoires, …) du joueur sont alors affichées.
2. Si le joueur appuie sur le bouton de retour, retourner au point 2 du scénario *Faire une* action en ligne.

Réglages :

Prérequis : Le joueur est authentifié, possède les droits d'administrateur et a choisi l'option *Réglages* du point 3 du scénario *Faire une action en ligne.*

1. L'administrateur a alors le choix entre :
    a. Modifier les scores -> suivre le scénario
    b. Modifier les scores :
    c. Modifier un mot de passe -> suivre le scénario *Modifier un mot de passe :*
2. Retourner au point 2 du scénario *Faire une action en ligne*.

Modifier les scores :

1. L'administrateur doit entrer le login du joueur dont il souhaite modifier le score.
2. Le serveur vérifie que le login existe bien dans sa base de données et renvoie une erreur si ce n'est pas le cas.
3. Si le login existe, le score du joueur est affiché.
4. L'administrateur peut en modifier la valeur.
5. La nouvelle valeur entrée est enregistrée par le serveur et remplace le score du joueur.
6. Retourner au point 1 du scénario *Réglages* :.

Modifier un mot de passe :

1. L'administrateur doit entrer le login du joueur dont il souhaite modifier le score.
2. Le serveur vérifie que le login existe bien dans sa base de données et renvoie une erreur si ce n'est pas le cas.
3. Si le login existe, l'administrateur peut entrer un nouveau mot de passe. Il ne peut toutefois pas voir le mot de passe actuel.
4. Le nouveau mot de passe entré par l'administrateur est attribué au joueur et enregistré par le serveur. Il remplace l'ancien mot de passe du joueur qui devra désormais utiliser le nouveau mot de passe avec son login pour se connecter lors de l'étape 2 du scénario *Se connecter* :.
5. Retourner au point 1 du scénario *Réglages* :.

Scénarios d'erreur :

Erreur de connexion (scénario Se connecter :) :

a. La connexion avec le serveur ne peut pas se faire.
   i. Un message signalant l'erreur est affiché.
   ii. Le joueur ne peut jouer qu'en local.
   iii. Les options *consulter ses statistiques* et *Ajouter un ami à sa liste d'amis* ne sont pas disponibles.
   iv. Les statistiques du joueur ne sont pas enregistrées.
   v. Le joueur peut cliquer sur *jouer en ligne* pour tenter une nouvelle connexion. Si elle fonctionne, reprendre le scénario *Se connecter* : au point 2.

Erreur d'authentification (scénario Se connecter :) :

2a. L'authentification a échoué.
   i. Un message signalant une erreur dans le login ou le mot de passe est affiché.
   ii. Le joueur est invité à ré-entrer son login et son mot de passe.
   iii. Si le joueur annule l'authentification, alors le joueur n'est pas connecté et ne peut jouer qu'en local.
   iv. Les options *consulter ses statistiques* et *Ajouter un ami à sa liste d'amis* ne sont pas disponibles.
   v. Les statistiques du joueur ne sont pas enregistrées.
   vi. Le joueur peut cliquer sur *jouer en ligne* pour tenter une nouvelle connexion.
   vii. Si l'authentification réussi, le scénario *Se connecter* : au point 2.

Un joueur se déconnecte (scénario Lancer une partie en ligne :)

a. Lors d'une partie en ligne (*1v1* ou *multi-joueurs*), si un joueur se déconnecte, il est immédiatement éliminé.
   S'il ne reste plus qu'un joueur, la partie est terminée et le joueur restant est déclaré vainqueur.
b. S'il reste plus d'un joueur (partie *multi-joueurs*), reprendre au point 7 du scénario *Lancer une partie multi-joueurs en ligne* :.

Base de données : Modèle conceptuel

Notre base de données est très simple, elle permet de stocker les utilisateurs pouvant se connecter en ligne afin de jouer des parties de Pong. Chaque utilisateur comporte :

- un *idUser* unique qui est auto-incrémenté.
- Un *username* et un *password* qui permet à l'utilisateur de s'authentifier afin de pouvoir se connecter en ligne.
- Son nombre de victoire *nbWins* ainsi que son nombre de parties jouées *nbPlays*.

- Un booléen qui définit si l'utilisateur est un administrateur et peut donc faire des modification sur les comptes d'autres utilisateurs (changer le mot de passe ainsi que le nombre de victoires et de parties jouées d'un autre utilisateur).

Conception du projet

Structure du projet

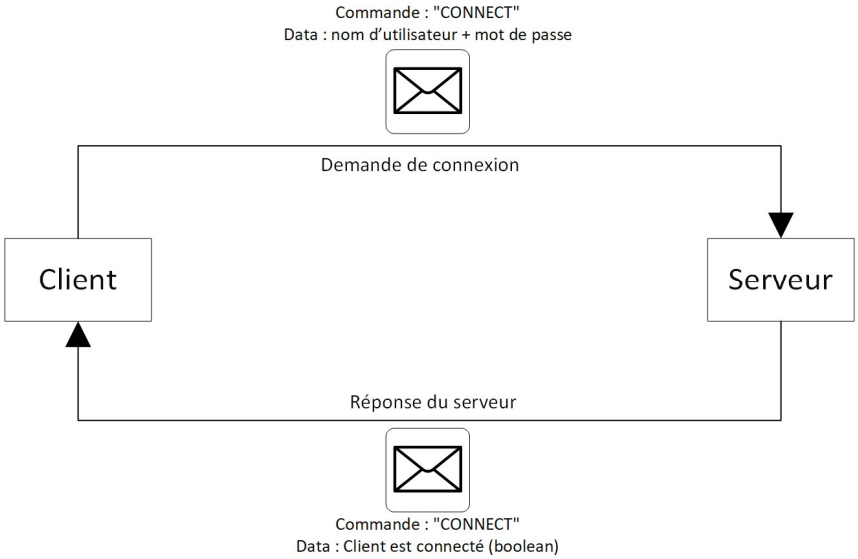
Dans notre projet, nous avons structuré notre projet en trois projets Maven :

- Un pour le client
- Un pour le serveur
- Un pour la libraire que nous avons appelé lib qui contient tous le code en commun entre le client et le serveur.

Le client et le serveur ont tous deux comme dépendance cette librairie lib. Cette manière de faire nous a permis d'avoir un code similaire du côté client et serveur sans avoir besoin de dupliquer le code.
Le problème est que les dépendance *Maven* doivent être compilé, donc à chaque changement dans les classes de la lib, il nous faut tous recompiler avec *Maven* ce qui prends un temps considérable, surtout lorsque l'on veut faire des tests.

Protocole d'échange entre le client et le serveur

Tous les échanges entre le client et le serveur se fait via une class nommé Protocole. C'est la seule classe qui sera sérialisée afin de permettre la communication entre le client et le serveur. Ce protocole contient deux champs, le premier étant le nom de la communication et le second le contenu aussi appelé *Data*.
Toutes les communications entre le serveur et le client se font de la même manière, nous prenons ici l'exemple de la commande "CONNECT" permettant de se connecter en ligne :



Le contenu est proprement parler du message *Data* est défini par une interface *IData*. Tous les messages que l'on voudra communiquer devront implémenter cette interface.
Le nom de la communication est un string et est déterminé par des éléments statiques dans la classe *SuperPongProtocole*.
Voici les différentes commandes de notre protocole que peuvent se transmettre le client et le serveur ainsi que les variables de *Data* pour le client et pour le serveur :

Commande	Data coté client	Data coté serveur
"CONNECT"	Son nom d'utilisateur et son mot de passe	Si la connexion a réussie
"SHOW_STATS"	Rien	Le nombre de Victoire et le nombre de parties jouées
"INSCRIPTION_GAME"	Son nom d'utilisateur, le nombre de joueurs pour sa partie et si la partie comporte des objets	Si le joueur a joint la partie et l'id du joueur pour cette partie
"PLAY"	Soi-même	Les joueurs en train de jouer, la balle, si le jeu est fini et si y a un item, l'item en question
"DISCONNECT"	Son nom d'utilisateur	Si la déconnexion a réussie (booléen)
"CHANGE_PASSWORD"	L'username et le nouveau mot de passe de l'utilisateur	Si le changement de mot de passe à réussi (booléen)
"CHANGE_STATS"	L'username de l'utilisateur à qui on veut changer les stats, son nouveau nombre de victoires et son nouveau nombre de parties jouées.	Si le changement des statistiques a réussi (booléen)
"USER_STATUS"	Rien	Si l'utilisateur est un administrateur (booléen)

Diagrammes de classes

Comme décrit lors du chapitre *Structure du projet*, par soucis de non redondance du code, nous avons créé un projet appelé lib qui regroupait les informations communes du client et du serveur. Après modélisation d'un diagramme de classe pour le client et un diagramme de classe pour le serveur, il s'est avéré que ces diagrammes étaient peu compréhensibles s'ils n'étaient pas reliés par la structure du projet lib. Un nouveau diagramme a alors été établit afin de comprendre les interactions entre les trois projet *Maven*. Ce diagramme étant volumineux, il se trouve en annexe. Dans ce diagramme, les classes des projets client, lib et serveur sont reparties selon leur appartenance à un projet. Des couleurs ont également été apportées afin de faciliter la compréhension du but de chaque classe.

- Les classes en vert sont les classes implémentant les différents menus de l'application. Ces différents menus sont gérés par la classe Displayer.
- Les classes en gris sont les classes qui définissent l'interface client d'une partie de Pong.

- Les classes en violet sont les classes définissant les vues des composants utiles au jeu.
- Les classes en bleu sont les classes définissant les composants du jeu qui sont utiles au client et au serveur.
- Les classes en rose sont les classes qui régissent de la communication entre le client et le serveur.
- Les classes en jaune implémentent la partie serveur de l'application.

Modèle relationnel de la base de données

Comme notre base de données est très simple et ne comporte qu'une seule table (voir *Base de données : Modèle conceptuel*), nous n'avons pas intégré de modèle relationnel à ce rapport.

Implémentation du projet

Technologies utilisées

Nous avons choisi de réaliser notre application en *Java* car c'est le langage où nous nous sentions tous le plus à l'aise pour implémenter cette application.

Pour l'interface graphique, nous nous sommes appuyés sur la librairie *JavaFX* d'Oracle. Cette librairie offre de nombreuses possibilités permettant d'obtenir rapidement des interfaces graphiques relativement élégantes. L'un des grands atouts de cette librairie est les fichiers *FXML* qui, avec des outils tels que *Scene Builder*, permettent de créer l'interface graphique visuellement.

Nous avons également utilisé *GitKraken* qui est une interface utilisateur de *git* pour *Windows*, *Mac* et *Linux*. Cette technologie nous a permis de travailler sur des branches différentes lorsque l'on voulait ajouter une nouvelle fonctionnalité et de fusionner cette fonctionnalité au projet principal dès qu'elle était terminée.

Nous avons aussi utilisé Docker ce qui nous a permis de simuler un serveur distant sur une machine locale et de faciliter le déploiement de notre application par la suite.

Problèmes rencontrés avec les technologies utilisées

L'apprentissage de *JavaFX* nous a pris beaucoup plus de temps que ce que l'on imaginait. Nous nous sommes souvent confrontés à des problèmes dont nous n'avions pas les connaissances nécessaires et qui ont nécessitées de longues recherches sur Internet, recherches qui débouchaient un résultat parfois frustrant par sa simplicité.

Concernant *GitKraken*, cet outil nous a parfois joué des tours lors de merge que nous avons fait. En effet, et ce surement à cause de mauvaises manipulations de notre part, certaines fusions ont eu pour effet de supprimer des parties de code qui furent totalement impossibles à récupérer par la suite. Nous avons donc du parfois réécrire du code à la suite de merge ou de push mal effectués.

La sérialisation nous a pris beaucoup plus de temps que prévu. Nous avons utilisé la librairie Jackson que nous ne connaissions que très peu. Il a fallu un certains moment pour la maîtriser et nous n'avions pas planifié cela. Nous avons souvent dû envoyé des sous-classes, ce qui nous a posé pas mal de problèmes et fait perdre du temps précieux.

Gestion du projet

Rôle des participants au sein du groupe de développement

Durant ce projet le groupe c'est divisé en 3 rôles :

- Le ScrumMaster : Ce rôle a été pris par Jérémie. Il consiste à l'organisation du projet, la mise à jour des bilans de sprint et la gestion de IceScrum.
- Le Protocol Master : Ce rôle a été pris par James. Il consiste à développer les protocoles de communication serveur et à les implémenter. Il a aussi fortement aidé à la réalisation des diagrammes.
- Les développeur : Ce rôle a été pris par Antoine et Benoît. Il consiste à la réalisation et au développement de l'application.

Nous avons assigné ces taches par rapport à nos intérêts en niveaux de connaissances. Mis à part le ScrumMaster qui a été choisi à la courte paille. Cependant, nous ne nous somme pas limités à la répartition des rôles. Par exemple, James a aidé le ScrumMaster à l'élaboration des histoires.

Backlog de produit :

Voir le fichier *Backlog Produit* fournit en annexe.

Les histoires sont regroupées en différentes couleurs selon la fonctionnalité à laquelle elles sont liées :

Suivi du projet :

Sprint n°1 – plan d'itération initial:

But du sprint : Planifier le projet et avoir une raquette qui bouge verticalement.
Durée du sprint : 2 points d'histoire
Histoires du sprint :

Bilan du Sprint n°1 :

Bilan sur la terminaison des histoires :

Les histoires n°1 et 18 étaient planifiées pour ce sprint. Elles ont toutes deux étés terminées dans le temps imparti. Par soucis de lisibilité et de facilité de compréhension, nous avons finalement réalisé un modèle de domaine plutôt qu'un diagramme de classe.

Vélocité du sprint :

Nous avons réalisé les 2 histoires planifiées pour ce sprint. La vélocité du sprint est donc de 2.

Replanification :

Nous sommes dans les temps selon notre planification actuelle. Il n'y a pas eu de modifications à ce niveau.

Rétrospective :

Le modèle de domaine a été plus complexe que prévu à créer et nous n'en sommes pas totalement convaincu. Plusieurs éléments risquent d'être modifiés en fonction de l'implémentation que nous déciderons de mettre en place.

Rétrospectives personnelles :

Nous avons tous pris le temps prévu afin de réaliser nos tâches pour cette itération. Jérémie a cependant dû consacrer quelques heures supplémentaires afin de réaliser les tâches administratives que nous n'avions pas prévu comme histoire dans la planification du sprint n°1.

Sprint n°2 – plan d'itération initial:

But du sprint : La balle doit pouvoir rebondir sur la raquette avec un certain angle. Le client doit pouvoir naviguer dans les différents menus.
Durée du sprint : 5 points d'histoire
Histoires du sprint :

Bilan du Sprint n°2 :

Bilan sur la terminaison des histoires :

Les histoires n°2, 3 et 4 étaient planifiées pour ce sprint. Elles ont toutes deux étés terminées dans le temps imparti.

Vélocité du sprint :

Nous avons réalisé les 3 histoires planifiées pour ce sprint. La vélocité du sprint est donc de 5.

Replanification :

Nous sommes dans les temps selon notre planification actuelle. Il n'y a pas eu de modifications à ce niveau.

Rétrospective :

La barre se déplace en saccader, il faudra voir pour résoudre ça au prochain sprint.

Rétrospectives personnelles :

Tout s'est bien passé.

Sprint n°3 – plan d'itération initial:

But du sprint : Le client doit pouvoir jouer contre une IA avec ou sans les objets.
Durée du sprint : 5 points d'histoire
Histoires du sprint :

Bilan du Sprint n°3 :

Bilan sur la terminaison des histoires :

Les histoires n°20 et 8 étaient planifiées pour ce sprint. Elles ont les deux étés terminées dans le temps imparti.

Vélocité du sprint :

Nous avons réalisé les 2 histoires planifiées pour ce sprint. La vélocité du sprint est donc de 3.

Replanification :

Nous sommes dans les temps selon notre planification actuelle. Il n'y a pas eu de modifications à ce niveau.

Rétrospective :

Il est possible de jouer correctement en 1v1 avec IA. Nous avons modifié la raquette et elle se déplace au mouvement de la souris. Elle n'est plus saccadée.

Rétrospectives personnelles :

Cette semaine fut compliquée car nous devions rendre le projet de semaine. C'est pour cela que nous avons été limite avec le temps. Mais toutes les tâches ont été exécutées.

Sprint n°4 – plan d'itération initial:

But du sprint : Créer le serveur ainsi que la gestion des utilisateurs. Un client doit pouvoir se connecter avec son compte en réseau.
Durée du sprint : 5 points d'histoire
Histoires du sprint :

Bilan du Sprint n°4 :

Bilan sur la terminaison des histoires :

Pour ce sprint, uniquement l'histoire 21 a été terminée. Les histoires 19 et 5 n'ont-elles pas été effectuée.

Vélocité du sprint :

Comme uniquement l'histoire 21 a été terminée, la vélocité du sprint est de 5.

Replanification :

Nous avons déplacé l'histoire 5 au sprint suivant (le n°5). L'histoire 19 sera potentiellement supprimée car c'est une histoire non prioritaire.

Rétrospective :

Le client possède une interface pour jouer en réseau, le serveur répond en affichant en console le log du client. La communication fonctionne donc entre le client et le serveur. Le

serveur se lance et crée des threads pour chaque client. La base de données ainsi que le contrôle de l'identification de l'utilisateur sont reportées au sprint suivant.

Rétrospectives personnelles :

Il est compliqué de créer le protocole de communication entre serveur car nous n'avons pas encore toutes les connaissances sur ce domaine. Nous remercions Miguel l'assistant du cours de RES pour les conseils qu'il nous a donné. Cette semaine a aussi été chargée en fonction des autres cours, notamment la présentation du projet de semestre qui a dû être effectué, ce qui peut expliquer le retard que l'on a pris.

Sprint n°5 – plan d'itération initial:

But du sprint : Le client doit pouvoir jouer en 1 contre 1 contre un autre utilisateur en réseau.
Durée du sprint : 5 points d'histoire
Histoires du sprint :

Bilan du Sprint n°5 :

Bilan sur la terminaison des histoires :

Pour ce sprint, nous avions l'histoire 6 à effectuer ainsi que l'histoire 5 que nous avions replanifiée à cette itération. Malheureusement, même si ces deux tâches ont bien avancé, aucune n'a été terminée à 100%. L'histoire 5 est réalisé à 80% tandis que l'histoire 6, elle, est réalisée à 20%.

Vélocité du sprint :

Comme aucune histoire n'a été terminée à 100%, la vélocité de ce sprint est de 0.

Replanification :

Nous avons replanifié les histoires 5 et 6 au sprint suivant (le n° 6).

Rétrospective :

La base de données est existante mais elle n'est pas reliée à la connexion. Du coup la connexion est validée dans tous les cas. Les informations sont bien reçues côté serveur. La déconnexion fonctionne, le serveur prend en compte la demande et supprime le joueur.

Rétrospective personnelle :

Bien que la vélocité de ce sprint soit de 0, ce sprint a servi à une nette amélioration au niveaux de la communication entre le client et le serveur. Il y a eu plusieurs problèmes avec Maven sur Linux. En plus de ce qui est indiqué, du temps a été passé pour la distribution (docker) du jeu. Beaucoup de temps a été passé à la sérialisation ce qui explique ce petit retard. Nous avons également signalé que le jeu final sera certainement limité à des parties 1 contre 1 en réseau, donc que les parties à plus de 2 joueurs ne seront pas implémentées.

Sprint n°6 – plan d'itération initial:

But du sprint : Le client doit pouvoir jouer contre d'autre joueurs dans une partie multi-joueur. Les scores des parties doivent pouvoir s'enregistrer dans les statistiques du client.
Durée du sprint : 7 points d'histoire
Histoires du sprint :

Bilan du Sprint n°6 :

Bilan sur la terminaison des histoires :

Comme annoncé lors du sprint précédent, l'histoire n° 7 a été abandonnée par faute de temps et pour pouvoir se concentrer sur d'autres tâches plus prioritaires. L'histoire n° 10 n'a pas été implémentée lors de ce sprint.
Les histoires n° 5 et 6 avaient été replanifiées à ce sprint. L'histoire n° 6 permettant au client de jouer contre un autre utilisateur a été terminée. L'histoire n° 5, elle, n'est toujours pas terminée car elle a été jugée moins prioritaire.

Vélocité du sprint :

Nous avons terminé l'histoire n°6 lors de ce sprint. La vélocité du sprint est donc de 5.

Replanification :

Le jeu en réseau avec plus que 2 joueurs est définitivement abandonnée. À la suite d'une mauvaise manipulation de notre part, la tâche est marquée comme étant faire sur IceScrum alors que ce n'est pas le cas. Nous n'avons pas trouvé le moyen de la supprimer.
Les statistiques (histoire n°10) et la connexion (histoire n°5) ont été replanifiées au sprint suivant (sprint n°7)
Une nouvelle histoire est apparue, celle des parties 1 vs 1 avec des items, qui est planifiée lors du sprint suivant (sprint n°7).

Rétrospective :

Nous avons bien avancé pendant ce sprint. En effet, nous avons réussi à jouer en 1v1 à l'aide d'un serveur sans qu'il y ait de lag, ce qui ne fut pas facile.
L'histoire n°6 nous aura pris énormément plus de temps que nous l'avions planifié initialement. Il manque juste le fait qu'il partie prend fin une fois qu'un joueur a atteint un certain score, ce qui est très rapidement implémenté.

Rétrospective personnelle :

Toute la gestion du protocole entre le client et le serveur aura été compliqué à mettre en place et la gestion d'un jeu en temps réel s'est avérée beaucoup plus ardue que l'on imaginait. Ceci explique en grande partie les retards que l'on a accumulé et donc l'abandon des différentes histoires que cela à entrainer.

Sprint n°7 – plan d'itération initial:

But du sprint : Le client doit pouvoir ajouter un ami à sa liste d'ami et jouer des parties avec ses amis. Un administrateur doit pouvoir modifier les statistiques des autres utilisateurs. Rédiger le rapport et que le manuel d'utilisation ainsi que d'effectuer des tests sur le programme.
Durée du sprint : 16 points d'histoire
Histoires du sprint :

Bilan du Sprint n°7 :

Bilan sur la terminaison des histoires :
Beaucoup d'histoires (trop d'histoires) avaient été planifiées lors de ce sprint. En plus de ces histoires planifiées s'ajoutent les histoires n°5 et 10 ont été replanifiées à ce sprint. En plus de ce histoires, nous avons ajouté l'histoire n°19 que nous avions laissé de côté lors du sprint n°4. A la fin de ce sprint, les histoires n°5, 10, 12, 15, 16, 17 et 19 ont été terminées.

Vélocité du sprint :
Avec ces 7 histoires terminées lors de ce sprint, la vélocité de ce dernier est de 16.

Replanification :
Il n'y pas de replanification possible. Cependant, les histoires 13 et 14 qui consistent à jouer avec un amis ont été supprimées. Ces histoires ainsi que le mode de jeu en multi-joueurs feront leur apparition dans la prochaine version de l'application, qui est prévue pour fin juillet lorsque l'équipe de développement se sera remise de ce semestre quelque peu angoissant.

Rétrospective personnelle :
Ce sprint fut le rush final. La grande vélocité de ce sprint s'explique par le fait que la plupart des histoires que l'on a terminées durant ce sprint avaient déjà bien été avancées lors des sprints précédents. Nous avons essayé de finir un maximum de fonctionnalités pour arriver au résultat de base que l'on s'était fixé en début de projet.

Stratégie d'intégration du code de chaque participant

Afin de pouvoir coder tous ensemble sur le même projet, nous avons décidé de faire une repo GitHub. Cela nous a permis de coder des fonctionnalités en parallèle. De plus, nous avons

utilisé des branches pour ne pas compromettre le fonctionnement du projet pendant l'élaboration de celui-ci.

Pour la gestion des dépendances, nous avons décidé d'utiliser Maven de Apache. Grâce à ce module, nous avons pu faire de classes contenue dans un projet commun entre le serveur et le client. Typiquement, il gère les dépendances des protocoles que nous avons créés.

Depuis de début du projet, nous avons également crée un groupe Telegram. Ce groupe nous a permis de partager nos connaissances et différents liens utiles que l'on avait trouvé. Cela nous a aussi permit de poser des questions à toute heure et d'avoir une réponse rapide pouvant être discutée.

Etat des lieux

Ce qui a été implémenté

Une grande partie des objectifs que l'on s'était fixés personnellement en début de projet ont été implémentées. Sur les 19 histoires que l'on planifiées, seules 3 n'ont pas été réalisées, il s'agit des parties avec des amis ainsi que des parties multi-joueurs (à plus de 2 joueurs).

Nous avons atteint l'objectif principal que l'on s'était fixé en début de projet. En effet, il est possible de lancer une partie de Pong en local contre une intelligence artificielle (ceci avec ou sans les items), ou alors de se connecter en réseau afin de pouvoir lancer une partie contre un autre utilisateur (ceci avec ou sans les items).

Toute la gestion des menus a été réalisée, elle comprend également les boutons pour les fonctionnalités que l'on n'a pas encore implémenté, comme les parties multi-joueurs avec plus de 2 personnes ou alors les parties contre des amis, ceux-ci étant désactivés dans la version actuelle de l'application.

Toute la partie communication entre le serveur et le client est fonctionnelle et évolutive. Elle permet d'ajouter de nouvelles fonctionnalités facilement. Cette partie nous a pris énormément de temps

Ce qu'il resterait à implémenter

Dans ce qu'il resterait à implémenter, il y aurait principalement le fait de pouvoir jouer contre plus que 1 personnes en ligne. On avait imaginé une sorte de mini-tournoi où chaque fois qu'une personne prenait un but elle était éliminée et le terrain était redimensionné en un nouveau polygone. Notre projet a été tout du long penser pour intégrer ce mode de jeu. En l'état actuel, un utilisateur pourrait s'inscrire à une partie multi-joueurs (entre 2 et 8 joueurs selon son choix) et une partie se lancerait lorsque le nombre requis de personnes se seraient inscrits à la partie en question. La logique du *matchmaking* est donc déjà implémentée pour les parties multi-joueurs. Il manquerait juste à définir la logique du jeu, avec un polygone où chaque côté correspond à un but qu'un joueur doit défendre.

Il nous resterait aussi à implémenter le fait de pouvoir ajouter de nouveaux amis parmi les utilisateurs de l'application et de pouvoir lancer des parties contre ses amis. Cette partie ne nous semble pas très compliquée à mettre en place. Nous l'avons finalement laissée tomber par faute de faute et car nous la trouvions moins prioritaire que certaines autres tâches qu'il nous restait à effectuer. Il suffirait d'ajouter une nouvelle table dans notre base de données qui définit quel utilisateur est ami avec qui (en prenant les id des utilisateurs).

Nous n'avons pas implémenté ces deux fonctionnalités car nous avons pris plus de temps pour concevoir et implémenter le serveur que ce qu'on avait imaginé initialement. Nous avons aussi eu certains bugs qui nous ont pris beaucoup de temps à résoudre. Comme mentionné lors du bilan du sprint 7, l'équipe de développement a planifié la prochaine version du jeu prenant en compte ces fonctionnalités, ceci si elle arrive à se remettre de ses émotions après un semestre éprouvant.

Une autre fonctionnalité à laquelle on a pensé en cours de projet, mais qui n'avait pas été planifiée, serait qu'un utilisateur puisse se créer un nouveau compte afin de se connecter en ligne. Mais sachant le succès planétaire que va connaitre notre jeu, nous avons préféré contrôler nous-même les utilisateurs pouvant se connecter en ligne. Seules des personnes ayant une importance très élevée ont le privilège d'avoir un compte et de pouvoir se connecter en ligne.

Auto-critique

Si ce projet était à refaire, nous passerions beaucoup plus de temps à la modélisation initiale de notre projet. Il aurait été préférable de planifier premièrement le protocole de communication entre le client et le serveur, cela aurait permis de savoir comment modéliser nos différentes classes coté client et coté serveur. En effet, selon la planification de nos sprints, nous avons d'abord créé une partie 1 contre 1 contre une IA en local. Ce n'est que fois que l'on a voulu créer les parties sur le réseau que l'on s'est rendu compte que notre code n'était pas adapté. Les objets que l'on devait communiquer entre le serveur et le client n'étaient pas définit ou mal implémentés pour répondre à nos besoins. Nous avons donc perdu un temps précieux à remodéliser un code existant afin qu'il réponde aux spécifications dont nous avions besoin.

Nous avons aussi réalisé que c'était ambitieux de réaliser une application en temps réel pour notre première application réseau. Il aurait été beaucoup plus simple de réaliser un jeux tour par tour. Au début, nous voulions faire une jeux totalement fluide en ayant un taux d'images par secondes élevé. Mais nous nous sommes vite rendu compte que cela n'était pas possible car la connexion n'était pas assez performante et cela entrainait de la latence. Ce qui rendait le jeux injouable. Nous avons donc décidé de réduire le taux de rafraîchissement entre le client et le serveur, le client faisant lui les déplacements de la balle de son côté et se met à jour périodiquement avec la position de la balle que le serveur lui envoie. Cela permet un bonne fluidité du jeu sans surcharger le réseau de communications entre le client et le serveur.

Concernant la planification des sprints, nous avons remarqué en cours de projet que certaines histoires avaient été planifiées beaucoup trop tôt. En effet, la méthode Scrum préconise des mettre les tâches importantes en premier, ce qui a parfois été mal fait de notre part lors de

notre planification initiale. On pense notamment à l'histoire n°5 pour la connexion d'un utilisateur en réseau en entrant son nom d'utilisateur et son mot de passe. Cette histoire avait été planifiée en sprint 4 alors qu'elle n'était vraiment pas prioritaire. On s'en est rendu compte durant le projet et cela explique que cette histoire a été replanifiée à plusieurs reprises pour finalement être faite en sprint 7, là où on aurait certainement dû la planifier à la base.

Stratégie des tests :

Notre stratégie de test constituait, premièrement à tester au fur et à mesure des Sprint le bon fonctionnement du code que nous produisons. Nous avons principalement réalisé ces tests en affichant des informations dans la console pour savoir si nos instances s'exécutent correctement. A la fin du projet, lors de la dernière itération, nous avons re-testé toutes notre application afin de pouvoir corriger les derniers bugs. Vu que nous avons pris un peu de retards, les tests finaux ont été fait n'ont pas été aussi poussé que prévu. Cependant,
le fait d'avoir testé notre application tout au long du développement à fait en sorte qu'il n'y ait presque aucune erreur lors des tests finaux. Pendant cette phase, nous avons implémenté quelques tests unitaire avec l'aide de JUnit. Tous ces tests, même s'ils ne sont pas conséquents, sont passé avec succès.

Pour que nos tests réussissent, il faut lancer un serveur de notre application en localhost sur le port 9090.

On a implémenté un test coté client qui vérifie que le serveur nous renvoie la bonne réponse si on lui envoie la combinaison d'un nom d'utilisateur avec son mot de passe. Le test implémenté teste avec le nom d'utilisateur « james » et le nom de passe « asdf ». Cette combinaison est de base dans la base de donnée fournie.

Nous avons également effectué des tests JUnit pour tester les accès à la base de données. Nous avons testé les accès en lecture pour la vérification des utilisateurs et la vérification si un utilisateur est administrateur. Les tests de modification ont été fait sur le changement de statistique, le changement de mot de passe, l'ajout de victoire et de défaite pour un utilisateur.

Nous avons donc tester toutes les différents accès à la base de données.

Conclusion :

Nous sommes vraiment content du résultat que nous avons obtenu. Même si ce fut laborieux, nous avons réalisé un jeux jouable et fun. Nous sommes quand même un peu tristes de ne pas avoir pu implémenter toutes les fonctionnalités. Ceci s'explique par une surcharge de travail durant tout le semestre et une sous-évaluation du travaille à fournir pour l'élaboration de ce projet. En effet, créer une communication client-serveur en temps réel fût beaucoup plus complexe que prévu. D'ailleurs, nous transmettons nos remercîments à Miguel Santamaria pour l'aiguillage qu'il nous a fourni afin de pouvoir élaborer certains protocoles de communication.

En ce qui concerne la méthode AGILE, nous avons pris conscience de son utilité. Cependant, nous pensons que cette méthode n'est pas adaptée pour des étudiant. Il est très difficile de prévoir les charges de travail qui arrivent au fur et à mesure du semestre. Ce projet fût trop court pour bien découvrir l'intérêt de cette méthode. Il aurait aussi été préférable qu'il n'y ait pas de rendu les 2 premières semaines afin de pouvoir concevoir une bonne architecture du projet.
Nous avons eu beaucoup de problème avec le site IceScrum. Nous avons eu beaucoup de problèmes pour bien visualiser les différentes histoires réalisées et l'avancement du projet.

Annexes :

Vous trouverez en annexe :

- Le backlog produit extrait depuis le projet créé sur le site *IceScrum*[2].
- Le repo GitHub sur lequel se trouvent tous les fichiers du projet est disponible à l'adresse : https://github.com/SmithHeig/SuperPong
- Le manuel d'utilisation
- Le diagramme de classe du client et du serveur

[2] Le projet est disponible à l'adresse https://cloud.icescrum.com/p/SUPERPONG/#/project