Projet d'année - Lombric à Brac

Ouriaghli Ismaël Dupuis Emma Renard Simon
Grimau Florent Stevens Quentin

29 mars 2020

Table des matières

Préface

Avant de commencer notre récit épique, nous tenons à remercier feu nos camarades qui nous ont quitté trop tôt durant cette aventure extraordinaire. Ils n'ont malheureusement pas survécu au péripéties sans relâche qui nous ont guettés à chaque *commit* lors de l'avancement du projet.

Alexis, notre cher et tendre menu designer sans qui cette aventure serait restée une simple idée.

Vassilena, pleine de motivation mais surchargée de travail. Courage pour ses autres projets.

Diego, sa présence aux réunions nous était chère, son humour et surtout, sa volonté de se battre.

1 Introduction

1.1 But du projet

L'objectif de ce projet est de créer un jeu d'artillerie et de stratégie multi-joueur en réseau du nom de "Lombric à brac", où chaque utilisateur devra se connecter au jeu afin de pouvoir créer ou bien rejoindre une partie. Il aura aussi la possibilité d'être spectateur d'une partie en cours.

Avant de jouer les joueurs pourront nommer leurs lombrics. L'hôte de la partie pourra également s'occuper de quelques paramètres tels que le mode de jeu (match à mort, match à mort par équipe) ou également le nombre de lombrics par joueur (8 maximum).

Une fois la partie lancée, les lombrics seront dispersés aléatoirement sur le terrain. La partie pourra ensuite commencer. Les joueurs joueront chacun à leur tour et auront la possibilité de faire différentes actions. Ils pourront déplacer leur lombric sélectionné ainsi qu'utiliser un outil leur permettant d'attaquer un autre lombric ou de se repositionner.

Pour qu'une partie se finisse, il faut qu'il ne reste que les lombrics d'une seule équipe sur le terrain, ce qui fera gagner cette dernière. Ou bien, que tous les lombrics soient morts, mettant fin à la partie par une égalité.

1.2 Glossaire

pseudo : Pseudonyme

Macro : nom donné à une valeur pour faciliter la lecture du code

bug : bogue

spawn : apparition

print : affichage en terminal d'un caractère unicode

Sprite : Objet destiné à être affiché

GUI : de l'anglais "graphical user interface", c'est une interface graphique

Hôte : Utilisateur qui a créé la partie, c'est lui qui peut gérer les paramètres

Replay : Revisionnage d'une partie jouée au préalable

Chat : Fenêtre de discussion entre entre deux joueur

Zoomer : Agrandir une image

1.3 Historique du document

Version	Auteur	Date	Description
0.1	Simon R., Emma D.	25/11/19	Besoin utilisateur
0.2	Alexis M., Ismaël O., Emma D., Simon R., Florent G., Quentin S., Vassilena S., Diego B.A.	26/11/19	Création du squelette
0.3	Alexis M., Ismaël O.	30/11/19	Besoins du système
0.4	Florent G., Quentin S.	08/12/19	Modules
0.5	Diego B.A	08/12/19	But du projet
1.1	Ismaël O.	24/02/20	Affichage
1.2	Florent G., Quentin S.	26/02/20	module serveur + diagrammes
1.3	Simon R.	28/02/20	Diagramme de classe + affichage de partie
1.4	Emma D.	27/02/20	Corrections besoins utilisateur + module client

2 Besoins utilisateurs

2.1 Exigence fonctionnel

Connexion

Au lancement du jeu, l'utilisateur a le choix entre se connecter et créer un compte. Il peut uniquement se connecter s'il a déjà un compte. À l'inscription, il doit choisir un pseudo qui n'existe pas. Le serveur vérifie que les informations pour se connecter sont correctes. Si oui, il met l'utilisateur au statut "en ligne", ce qui lui permettra de recevoir des invitations, de chercher une partie, de communiquer avec ses amis, etc.

Menu principal

Une fois connecté, l'utilisateur accède au menu principal dans lequel il dispose de plusieurs choix. Il peut chercher une partie, en créer une, discuter avec des amis, accepter une invitation, renommer ses lombrics, gérer sa liste d'amis, consulter son historique, consulter le classement, revoir une partie ou se déconnecter.

Trouver une partie

Lorsque l'utilisateur souhaite trouver une partie pour jouer, il est mit en liste d'attente par le serveur. Ce dernier fera des vérifications récurrentes pour vérifier si une partie est disponible, et ainsi envoyer le joueur en attente vers cette partie.

En jeu

Durant la partie, l'utilisateur doit pouvoir déplacer ses lombrics, choisir une arme en contrôlant la puissance ainsi que l'angle, voir la vie de chaque lombrics a l'aide d'une barre de vie au dessus des lombrics (gui) ou sur un tableau des scores (terminal), pouvoir passer son tour. Le client doit aussi voir tout mouvement/tir/changement d'armes du joueur qui est en train de jouer.

Diagrammes

Des diagrammes UML sont utilisés pour décrire les différentes exigences fonctionnelles de l'utilisateur. Ces exigences sont séparées en 3 diagrammes distincts. Ils représentent respectivement les interactions d'un utilisateur dans le menu principal, dans le salon d'attente et dans une partie.

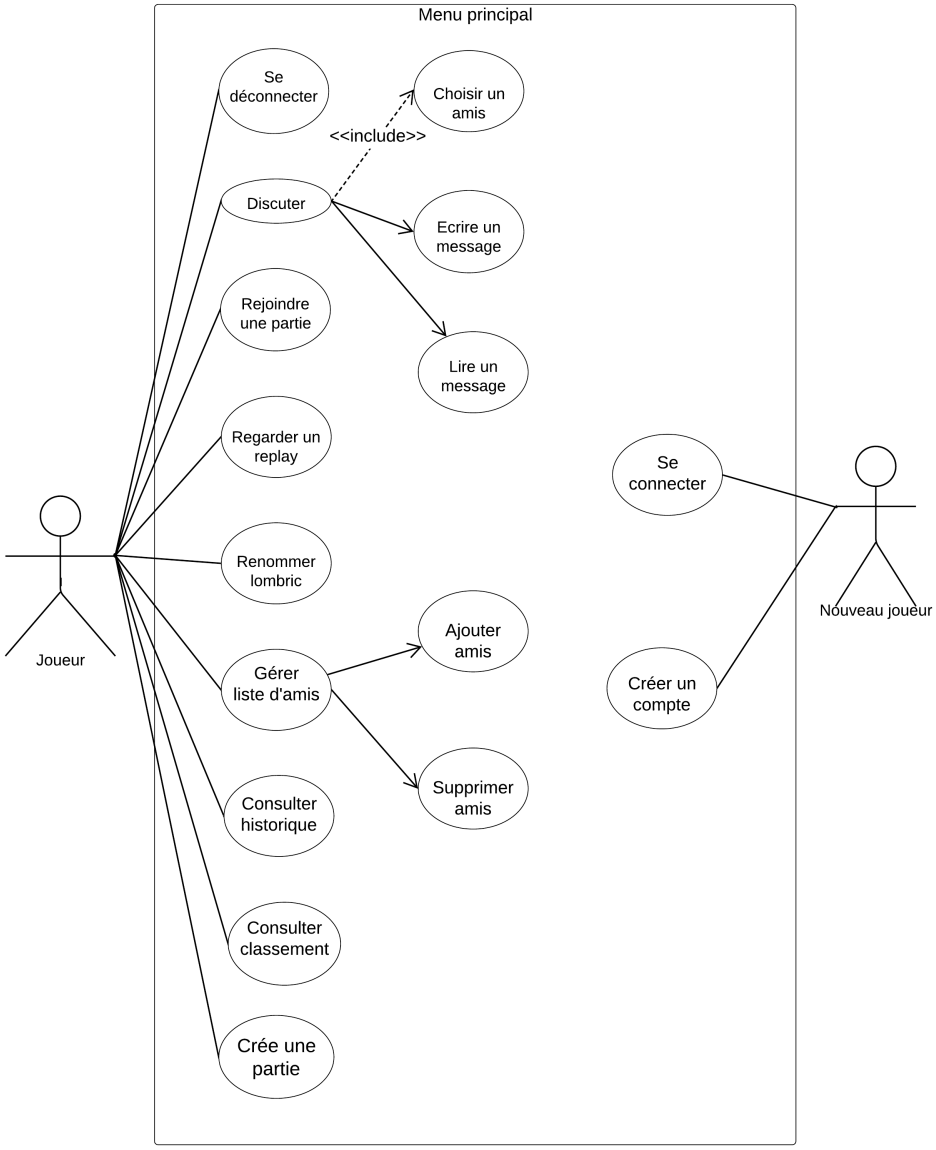

FIGURE 1 – Diagramme use case - Menu principal

Toutes les actions autres que "se connecter" et "créer un compte" impliquent que l'utilisateur est déjà identifié. Nous considérons une personne comme étant un joueur lorsque que celle-ci s'identifie. Elle a donc accès aux fonctionnalités du jeu. Dans le cas contraire, elle ne peut que se connecter ou créer un compte.

	Pré-conditions	Post-conditions	Cas général	Cas exceptionnels
Se connecter	— Avoir un compte — Être déconnecté	Le joueur est connecté	L'utilisateur rentre un pseudo et un mot de passe corrects	— Le joueur n'est pas connecté au serveur — Le pseudo ou le mot de passe est incorrect
Se déconnecter	Être connecté	— L'utilisateur est déconnecté — L'utilisateur est ramené à la fenêtre de connexion	L'utilisateur se déconnecte	L'utilisateur ferme la fenêtre sans se déconnecter
Créer un compte	L'utilisateur n'est pas connecté	Le joueur est connecté sur son nouveau compte	Le nom d'utilisateur n'existe pas encore	Le nom d'utilisateur est déjà pris
Ajouter amis	L'utilisateur n'a pas encore en ami la cible	L'utilisateur est ami avec la cible	Le nom cible est un nom d'utilisateur existant	Le nom de l'utilisateur cible n'existe pas
Supprimer un ami	L'utilisateur a choisi un de ses amis	L'utilisateur n'est plus ami avec la cible	L'ami est supprimé de la liste de l'utilisateur	Pas d'amis à supprimer
Regarder un replay	Il faut avoir au moins un replay	None	Le replay est en train d'être regardé	Le replay est corrompu
Renommer un lombric	— Un lombric a été choisi — Un nom a été choisi	None	Le lombric choisi a été renommé	Le lombric n'avait pas encore été nommé
Consulter historique	Un joueur est sélectionné	None	L'historique des parties est affiché	Le joueur sélectionné n'existe pas
Consulter le classement	None	None	Le classement est affiché	None
Recevoir un message	None	None	Le message est affiché	None
Envoyer un message	— Un ami est choisi pour recevoir le message — Un message est écrit	None	Le message est envoyé	None

FIGURE 2 – Tableau du use case du menu principal

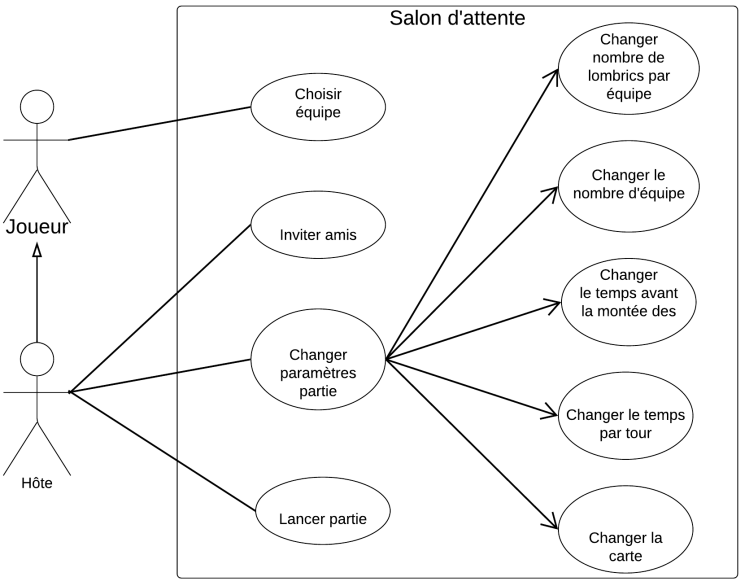

FIGURE 3 – Diagramme use case - Salon d'attente

Toutes les actions du salon d'attente impliquent que la partie n'a pas encore été lancée.

	Pré-conditions	Post-conditions	Cas général	Cas exceptionnels
Choisir une équipe	Équipe pas encore formée	Équipe formée des lombrics choisis	Joueur choisi ses lombrics	None
Changer les paramètres d'une partie	None	La partie est configurée	Le joueur choisit les paramètres désirés	None
Inviter des amis	— Avoir des amis — être l'hôte	Amis à rejoint la salle d'attente	Le joueur veut jouer avec un ami	Le joueur ami n'existe pas
Lancer la partie	— Les paramètres de la partie sont configurés — Le joueur à créé son équipe — Au moins deux joueurs ont rejoint la partie	La partie est lancée	Le joueur configure la partie et veut commencer à jouer	Il n'y a pas assez de joueurs pour lancer une partie

FIGURE 4 – Tableau du use case du salon d'attente

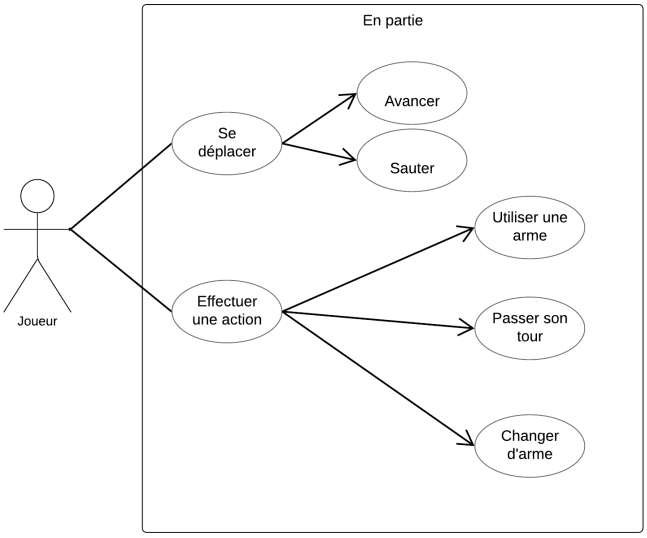

FIGURE 5 – Diagramme use case - Partie de jeu

Toutes les actions autres que discuter s'effectuent uniquement lors du tour du joueur.

	Pré-conditions	Post-conditions	Cas général	Cas exceptionnels
Se déplacer	Déplacement choisit est accessible	— Le joueur se trouve à une autre position — Le tour du joueur est fini	Joueur veut se déplacer sur le terrain	Si le joueur sort du terrain, il meurt
Utiliser une arme	— Avoir un lombric capable d'attaquer — Avoir au moins une arme	— Perte de point de vie de l'adversaire attaqué — Le tour du joueur est fini	Le joueur veut infliger des dégâts à l'adversaire	None
Passer son tour	None	Le tour du joueur est fini	Joueur à fini son tour	None

FIGURE 6 – Tableau du use case d'une partie en cours

2.2 Exigence non fonctionnelles

Le langage imposé pour ce projet est le c/c++. Le jeu doit au minimum tourner sur les ordinateurs de l'ULB (sous Linux).

En cas de perte de connexion d'un des joueurs, il faut faire en sorte que le cas soit traité étant donné que d'autres joueurs peuvent attendre ce dernier. Lorsqu'un joueur se déconnecte dans le salon d'attente, celui-ci est supprimé du salon. Lorsqu'un joueur se déconnecte en partie, celui-ci perd automatiquement la partie.

Le jeu doit pouvoir se jouer soit avec une interface graphique soit en console. Ces 2 versions devant être compatibles. Un joueur en console doit pouvoir jouer une partie avec un autre joueur qui utilise la GUI.

La GUI doit être agréable et simple à utiliser. L'utilisateur doit pouvoir avoir accès à toutes les informations sans difficulté.

2.3 Exigences du domaine

3 Besoins du système

3.1 Exigences fonctionnelles

Protocole de Communication Client/Serveur

En utilisant la librairie protocol-buffers, l'envoi et la réception de données est simplifié. Nous avons créé plusieurs prototypes, correspondants chacun à des actions particulières. Chacun de ces prototypes est associé à une *macro*, qui est utilisée pour spécifier quel prototype est envoyé par le réseau, et ainsi pouvoir le récupérer correctement de l'autre côté.

Création de compte et gestion de compte

Au lancement du jeu, le programme propose à l'utilisateur de soit créer un compte, soit de se connecter à un compte déjà existant. Si l'utilisateur décide de créer un compte, le programme affiche un formulaire d'inscription demandant le nom du nouveau compte (pseudo) ainsi que le mot de passe qui lui sera associé. Une fois son compte créé, lors des prochains lancements du jeu, l'utilisateur n'aura alors plus qu'à entrer son pseudo et son mot de passe afin de se connecter.

Création de son équipe de lombrics

Lors de la création du compte, une équipe de 8 lombrics est créée automatiquement avec des noms par défaut. L'utilisateur aura la possibilité, dès lors qu'il sera connecté, de changer le nom de ses lombrics, pour les remplacer par ce qu'il souhaite (Et ce autant de fois qu'il le voudra).

Jouer une partie

Création et configuration d'une partie : Le joueur crée une partie en choisissant les différents paramètres de cette dernière, qu'il pourra modifier par la suite. Une fois dans le salon, il peut choisir dans quelle équipe il souhaite être, ce qui permet de faire du 1v2 2v2 1v1v1v1 etc.

Gestion des invitations : Le joueur peut aussi inviter des personnes faisant partie de sa liste d'amis afin de jouer avec eux. Un joueur doit être connecté pour recevoir une invitation.

Trouver une partie : Le joueur va rejoindre une partie qui n'a pas encore été lancée par l'hôte. Si plusieurs joueurs sont en attente, la priorité ira à celui qui est en attente depuis le plus longtemps.

Gestion de la liste d'amis

Le joueur peut envoyer une demande d'ami à un autre joueur. Dès lors que ce dernier accepte l'invitation, ils deviendront amis et pourront alors s'envoyer des invitations de jeu et des messages privés.

Gestion messages privés entre amis

Si deux joueurs sont amis, ils peuvent discuter ensemble à l'aide d'un *chat*.

Consultation de l'historique de parties d'un joueur

Un joueur peut consulter l'historique d'un autre joueur ou le sien ainsi que son classement dans chacune des parties effectuées. Il dispose pour cela d'une fenêtre où il pourra entrer le pseudonyme du joueur dont il souhaite voir les parties.

consultation du classement des joueurs

Un joueur connecté peut consulter le classement global des 15 meilleurs joueurs, triés selon leur nombre de victoires respectifs.

3.2 Exigences non fonctionnelles

Besoin d'une connexion internet

Puisque le jeu se joue en réseau, une bonne connexion internet (faible latence) est requise afin de communiquer efficacement avec le serveur.

Mise à jour de la base de donnée

Le programme doit pouvoir enregistrer des (nouvelles) données comme le profil de l'utilisateur (son mot de passe, pseudo), son nom d'équipe, sa liste d'amis ou encore les noms de ses lombrics.

Gestion des erreurs

Quand le programme rencontrera une erreur, il ne s'arrêtera pas brusquement et gérera l'erreur en question comme il se doit.

Fluidité

Le programme doit être optimisé afin que pendant une partie il n'y ai que peu de latence et que la connexion entre l'utilisateur et le serveur ne soit pas lente. Le client et le serveur font chacun les calculs d'affichage pour que le jeu soit le plus fluide possible. Une fois un tour terminé, le serveur impose sa vision de la partie à tous le monde pour éviter les problèmes de désynchronisation ou de triche.

Optimisation

Il faut veiller a ce que le serveur ne soit pas trop gourmand en ressources (en évitant par exemple toute fuite de mémoire), pour que l'on puisse accueillir le plus grand nombre de joueurs possible sans risquer une surcharge de la machine qui exécute le serveur.

Anti-triche

Le serveur doit faire les calculs considéré important (trajectoire, dégâts) pour éviter toute triche potentielle. C'est lui qui aura le dernier mot sur l'état de la partie à la fin d'un *round*. Le client reçoit les informations pour tirer et reçoit, durant l'animation, les résultats du serveur. Si ces deux informations sont différentes, les résultats du serveur seront prioritaires, et le client oubliera ce qu'il a calculé.

Sécurité

Le serveur va hasher le mot de passe avant de les enregistrer dans la base de données pour éviter qu'une personne tierce ayant accès au serveur (machine), ait accès aux mots de passe (en clair) des utilisateurs, ce qui pourrait être un soucis dans le cas où ce dernier ait utilisé le même mot de passe ailleurs.

3.3 Design du système

Cette section montre comment le code à été pensé et réalisé a l'aide des différents diagrammes de classe.

Interface console

FIGURE 7 – Diagramme de classe - interface console

Interface graphique

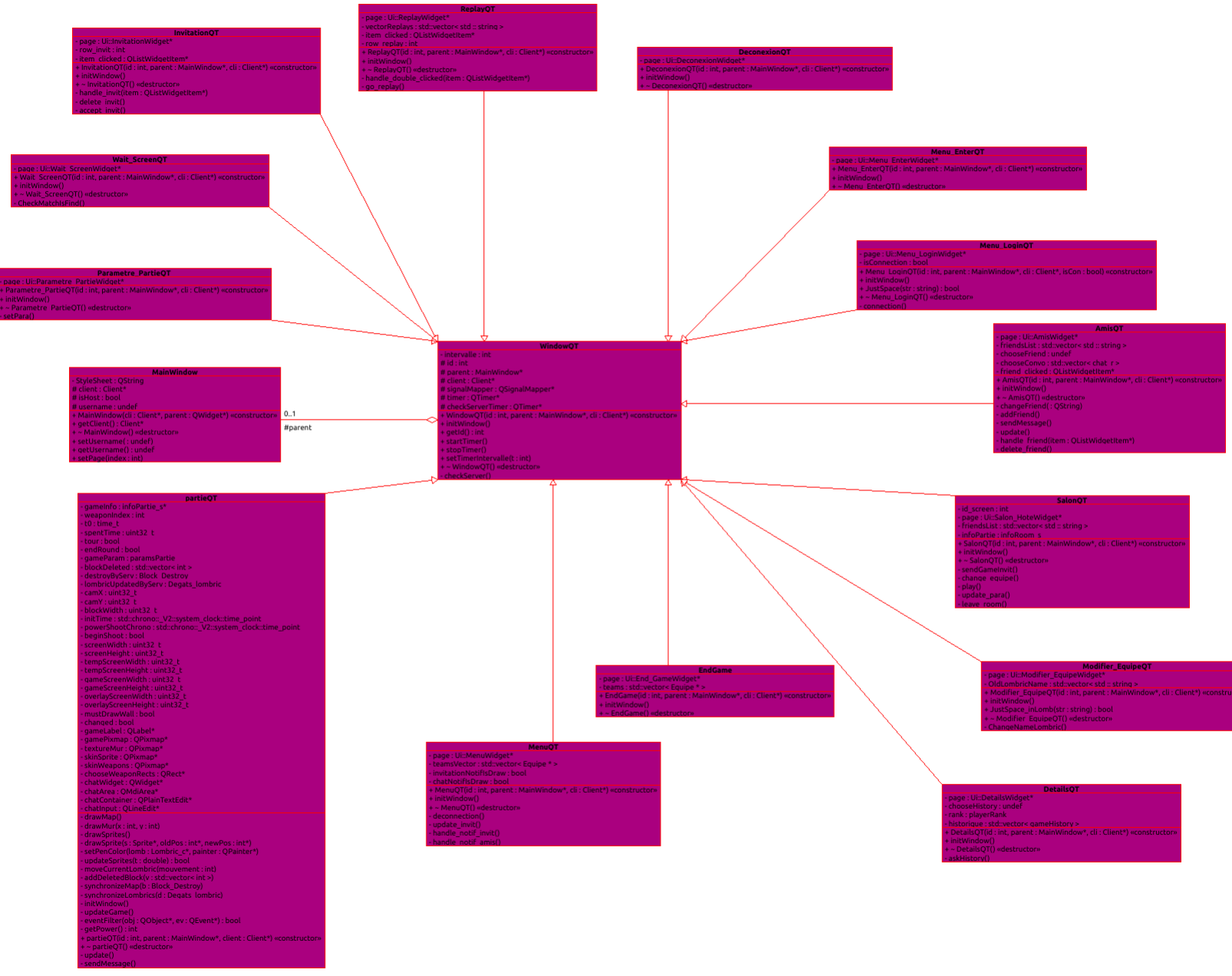

FIGURE 8 – Diagramme de classe - interface graphique

Gestion de la partie

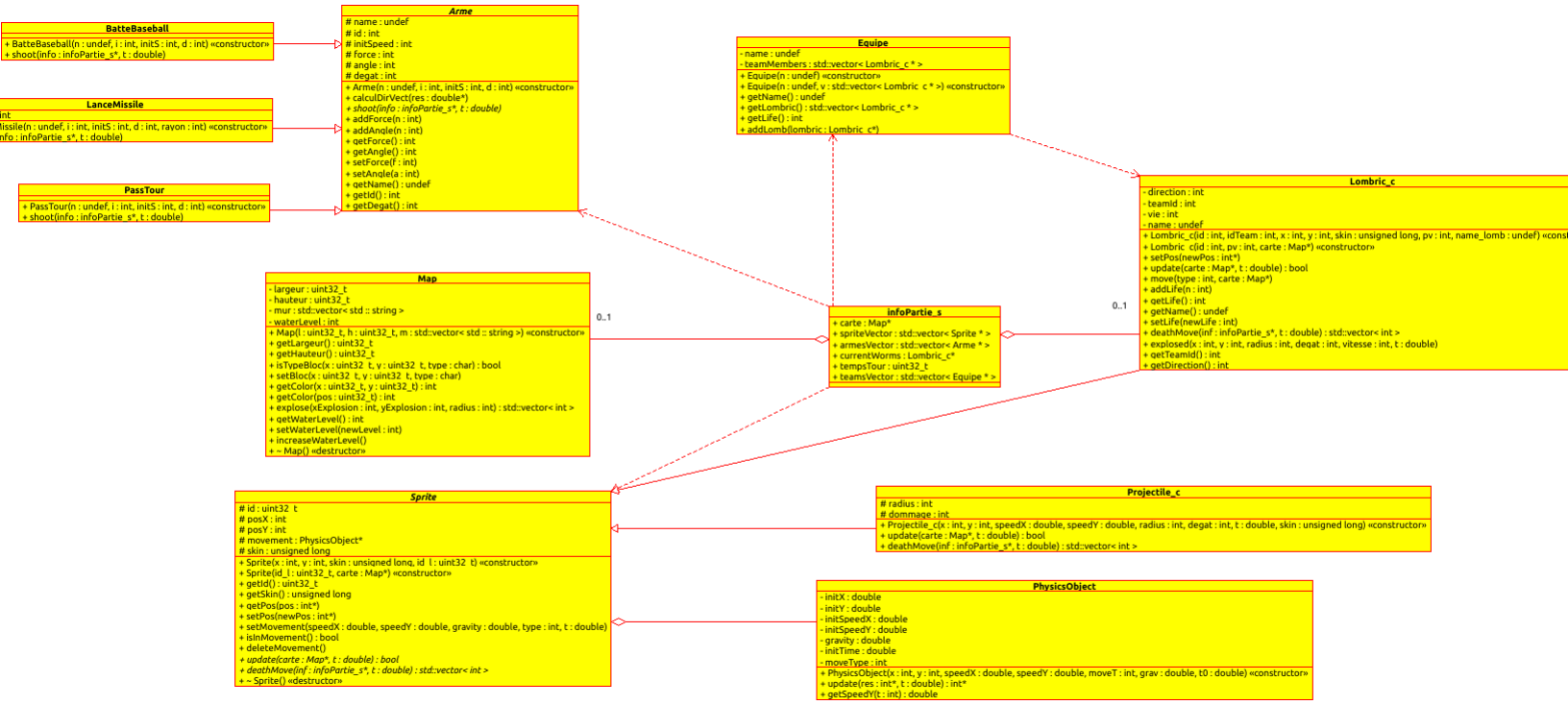

FIGURE 9 – Diagramme de classe - partie

Vue d'ensemble

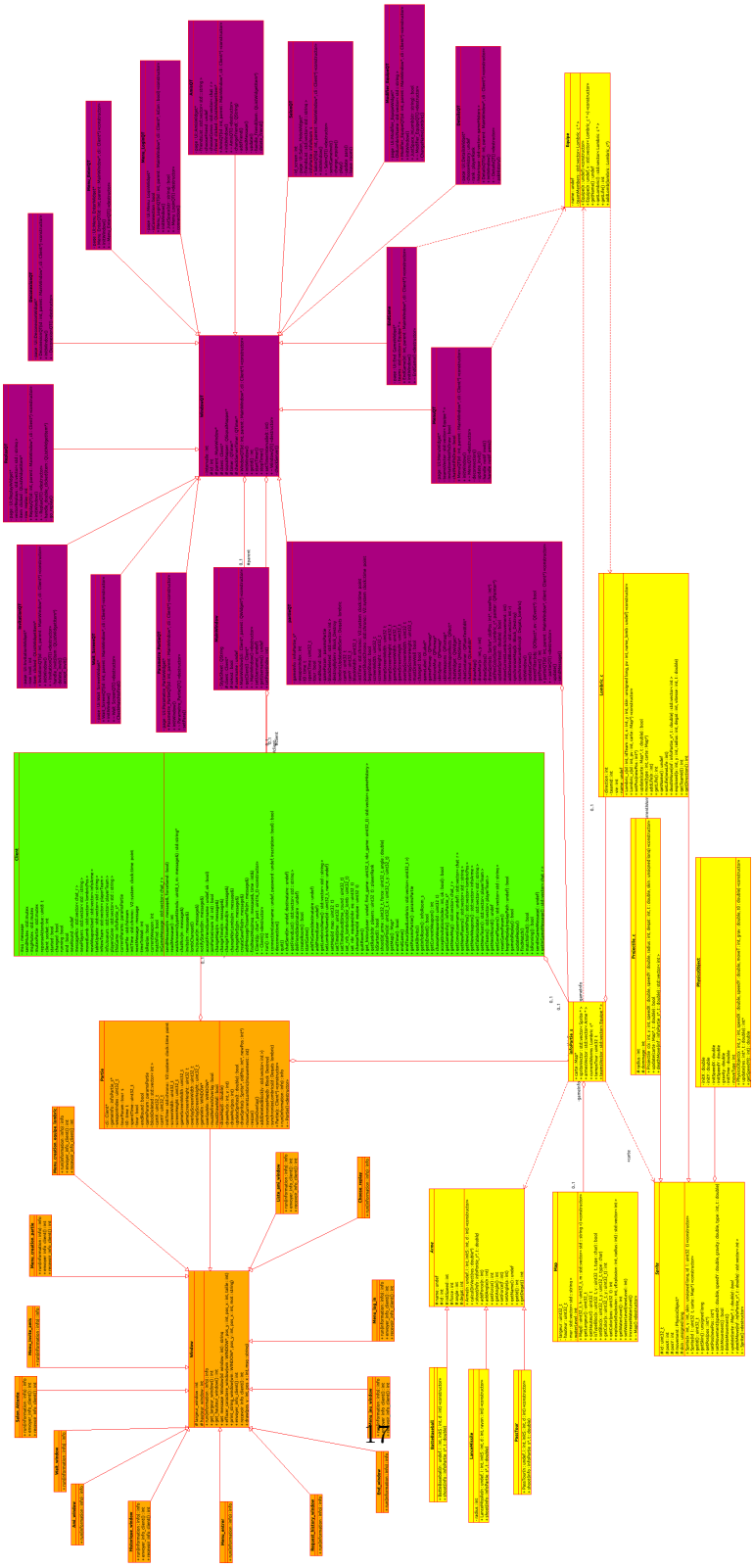

FIGURE 10 – Diagramme de classe - vue d'ensemble

4 Modules

4.1 Module Client

Dépendances

Ncurses : Cette librairie est utilisée pour l'affichage en console.
Qt5 : Cette librairie est utilisée pour l'affichage en interfaces graphique.

Fonctionnement des menus

Pour naviguer entre les différents menus nous utilisons une variable *res* qui contiendra l'*id* de la fenêtre à ouvrir. Pour ce faire, nous avons prédéfini une liste des différentes fenêtres avec leur *id* respectif et chaque fenêtre, retourne l'*id* de la prochaine fenêtre à afficher. Donc le main est juste une boucle infinie qui appelle la bonne fenêtre (le bon objet) en fonction de ce que contient *res*.

Menu entrer

Elle correspond à la première fenêtre auquel l'utilisateur sera confronté. Il aura trois choix devant lui :

— Se connecter
— S'enregistrer
— Quitter

Il pourra donc se connecter ou créer un compte. Si il y a un quelconque problème avec le pseudo ou le mot de passe entré, une fenêtre avec un message d'erreur approprié s'affichera et le redirigera sur le Menu Entrer.

Menu Jeu

Ici, l'utilisateur est dans le grand menu du jeu. Il aura plusieurs choix qui lui seront proposés (voir ci-dessous) et pourra effectuer différentes actions, telles que se déplacer dans le menu et dès qu'il appuiera sur la touche de confirmation, une nouvelle fenêtre s'affichera avec les bons éléments.

— Trouver partie
— Créer une partie
— Créer/Modifier équipe
— Voir ses invitations
— Classement
— Amis
— Historique
— Se déconnecter

Créer une partie

En appuyant sur créer une partie dans le menu jeu, l'utilisateur sera invité à entrer les différents paramètres de la partie dans la nouvelle fenêtre qui s'est affichée. Des messages d'erreur seront affichés s'il ne respecte pas les restrictions imposées (par exemple si il donne en input un *string* au lieu d'un *int*, ou bien si le *int* entré dépasse la limite autorisé).
À savoir, pour pouvoir récupérer l'entrée utilisateur dans la *box*, nous utilisons la fonction *mvwgetnstr()* de *ncurses* qui enregistre les entrées dans un tableau de char. Après avoir réceptionné le message, la fonction *stoi()* est utilisée pour transformer les entrées en *int* pour pouvoir tester la valeur entrée. Un *catch* de *std : :invalid_ argument* est effectué après le *stoi()* (dans le cas où un *string* est entré au lieu d'un *int*).
Après cela, l'utilisateur aura officiellement créé une *room* et se trouvera dans un salon d'attente où il verra les autres utilisateurs connectés à sa *room*. Dans le salon d'attente il peut soit :

— Annuler
— Options (changer les paramètres de la partie)
— Inviter un ami (dans le salon)
— Lancer la partie

Invitation et Ami

À plusieurs endroits, l'utilisateur peut interagir avec les invitations reçues et ses amis.
Dans un premier temps, dans le Menu Jeu en appuyant sur *Voir ses invitations* il pourra voir les invitations reçues (une distinction est faite entre invitation de jeu et une demande d'ami). Il peut soit accepter l'invitation (si il s'agit d'une invitation de jeu, il est alors redirigé dans le salon d'attente), soit renier l'invitation ou ne rien faire du tout.
Dans un second temps, dans l'onglet *Ami* (à savoir qu'il y aura une notification si l'utilisateur a un message non lu qui, si oui une astérisque s'affichera *Ami (*)*), l'utilisateur aura le choix entre :

— Envoyer demande ami
— Consulter liste amis
— Discuter avec ses amis
— Revenir au menu

Si le joueur veut envoyer une demande d'ami, il doit écrire dans la *box* indiquée, le pseudo de la personne à qui envoyer l'invitation. La deuxième option est de consulter sa liste, cela veut dire qu'il pourra voir ses amis et/ou en supprimer un si il le souhaite (à noter qu'il faut au préalable mettre la flèche devant l'ami à supprimer en appuyant sur *q*). Et enfin la dernière option, mais non des moindres, le chat. Dans cette option, il pourra choisir un ami à qui envoyer un

message, cette étape effectuée, une fenêtre s'affichera avec les 5 derniers messages échangés et une box où sera écrit le message. De plus en appuyant sur *r* l'utilisateur pourra *refresh* la fenêtre pour voir si des nouveaux messages ont été reçus.

4.2 Affichage de la partie

Gestion de la partie

Durant la partie, le joueur peut déplacer la caméra si la console ou la fenêtre est plus petite que la taille de la carte. Quand c'est à son tour, la fenêtre qui contient les informations de la partie change pour montrer quelle arme est sélectionnée et comment elle est paramétrée. Il voit également le temps restant pour tour.

En GUI, le joueur à également la possibilité de zoomer dans la carte du jeu.

Fonctionnalité

Chaque *sprite* peut subir un mouvement, soit parabolique, soit rectiligne uniforme. Les explosions font des dégâts circulaires sur la carte et expulsent les lombrics touchés par ces dernières. Quand un lombric est soumit à un mouvement, il ne peut pas être contrôlé par le joueur.

4.3 Communication avec le serveur

Le module client s'occupe de faire la passerelle entre l'affichage et le serveur. Il reçoit les messages du serveur et crée à partir de ces messages des structures adéquates pour l'affichage.

Thread

Le module client est un thread secondaire.

Le client réceptionne des messages dits inattendus (qu'il n'a pas expressément demandé au serveur) et les stocke dans des vecteurs jusqu'à ce que l'affichage les récupère.
L'affichage récupéra ce vecteur et le client pourra alors le vider. C'est le cas des messages du chat, des invitations d'amis et des invitations aux parties.

Le client réceptionne également des messages qu'il a expressément demandé au serveur. L'affichage aura donc fait appel précédemment à une méthode du client qui va interroger le serveur puis répondre à l'affichage.
Lorsque l'affichage appelle une de ces méthodes, il passe en paramètres les informations dont le serveur a besoin pour traiter la demande. Il s'agit des valeurs reçues en input et d'une macro indiquant au serveur de quel type de demande il s'agit.
Le client convertira alors une structure contenant ces informations en *string* et

les enverra au serveur.
S'il s'attend à une réponse, il se bloquera jusqu'à la réception de cette réponse. Une fois débloqué, il reconvertira le string reçu avec les informations demandées par l'affichage en un objet *proto-buff* qu'il transformera ensuite en une structure utilisable par l'affichage.
S'il n'attend pas de réponse, il se bloquera juste le temps de l'envoi du message. C'est le cas quand un tir est effectué. L'affichage envoie les paramètres du tir mais n'attend pas de réponse.
Les méthodes du client sont donc bloquantes pour s'assurer de la réception de la réponse (en cas de besoin). L'affichage ne peut rien faire entre sa demande et la réponse qui lui est donnée. C'est le cas lorsqu'il faut afficher le nom des lombrics ou encore lorsque l'utilisateur se connecte.

Librairie utilisée

Afin d'envoyer des messages structurés entre le serveur et le client, nous utilisons la librairie protocol-buffers. Cette librairie permet de définir des structures qui, après compilation, donnent une classe en $C++$. Cette classe permet de remplir les champs de la structure et peut simplement être convertie en *string*. Ce *string*, une fois envoyé au serveur peut être reconverti en la classe correspondante. Cette librairie est donc également utilisée par le serveur.

4.4 Module serveur

Dépendances

Le serveur utilise les librairies suivantes :

- Google's Protocol Buffers : Une librairie utilisée pour facilement transférer des données au travers du réseau, comme expliqué dans le module client.
- Sqlite3 : Une librairie utilisée pour stocker des données à long terme sur le serveur, de les récupérer et les trier facilement.
- Bcrypt : Une librairie de hachage de mots de passe, qui utilise l'algorithme Blowfish.
- ZeroMQ : Une librairie qui permet la communication inter-threads, au moyen d'un système de "publisher/subscriber", qui permet à des threads de s'abonner à des canaux de communication, ou à y envoyer un message.

Découpe en threads

La serveur est découpé en plusieurs threads :

— Thread principal : Ce thread est à la base de tous les autres, c'est celui qui attend des connections de la part des clients, et qui génère tous les autres threads. Dès qu'un client se connecte au serveur, il se réveille et assigne un thread (expliqué au point suivant) à ce client.

— Thread client : Un thread créé par le thread principal qui gère la communication avec un seul client à la fois. Il peut notamment interagir avec la base de donnée, lorsque l'utilisateur souhaite envoyer un message, récupérer ses messages reçus, voir sa liste d'amis, le classement général, son historique de partie, ou encore modifier les noms de ses lombrics.

— Broker : Le broker est un thread qui permet la communication entre les threads clients, ou entre les threads client et leurs salons respectifs, au moment de la création de ce dernier (après cela, les clients communiquent directement avec le salon, sans passer par le broker). Il agit comme une sorte de service postal, il reçoit des messages de tout le monde, qui sont soit destinés à un autre utilisateur, auquel cas il les redirige, soit destinés au broker lui même (Par exemple, un utilisateur peut demander au broker de créer un thread partie, ou autre). Ce thread est lancé au lancement du serveur par le thread principal, avant que celui-ci ne se mette en attente de connexion des clients.

— Partie : Un thread qui gère l'entièreté d'une partie, la communication avec les joueurs connectés à cette partie, le salon d'attente (avec modification des paramètres de partie, invitation de joueurs, ...)

Module Communication

Le module communication du serveur est représenté dans ce projet par une classe : "Listener". Cette classe possède plusieurs méthodes, en rapport avec la manière dont le client et le serveur communiquent. Elle permet entre autres de récupérer le type de message qui va arriver, et du préparer l'environnement à la réception du message en lui-même. Ce module permet également d'envoyer des messages, en précisant encore une fois le type de message que l'on désire envoyer.

Module Salon/Partie

Ce module est géré par un thread à part qui est lancé par le Broker lorsqu'un client souhaite lancer une partie. Le thread qui gère ce module est une sorte de mini broker, qui réceptionne tous les messages que les clients lui envoie, et les transfère aux autres clients, ou les interprètent. Il est donc abonné à son propre canal ("room/<id>/client") pour les messages de la part des clients sur la room <id>.

Le module salon gère l'attente des joueurs avant la partie. Il permet au modérateur de la partie (celui qui l'a créée, de modifier les paramètres de la

partie qui suit. Un salon contient des équipes, lesquelles contiennent des joueurs. Ces derniers possèdent leur équipe de lombrics. C'est durant cette phase que les joueurs peuvent en inviter d'autres à jouer.

Le module partie représente une partie jouée. Le module gère la carte, les interactions entre la carte et les joueurs le temps alloué à chaque joueur pour jouer son tour et l'apparition de bonus. Il s'occupe aussi de calculer tout ce qui est trajectoire des missiles ainsi que des dégâts pour éviter que le client puisse tricher sur les informations envoyées au serveur.

Module Base De Donnée

Ce module gère les interactions avec la base donnée, notamment lorsqu'un joueur s'inscrit, ou qu'une partie se termine. Il permet aux autres modules de faire des requêtes pour trouver des joueurs/parties/..., ou d'écrire dans la base de donnée au moyen de méthodes propres. Pour éviter les soucis de concurrence dans l'écriture en base de donnée, ce module, représenté par un objet, est déclaré globalement, et chacun peut y accéder, mais, via un mutex, on peut garantir qu'à aucun moment il n'y aura deux partie du code qui utiliseront cet objet en même temps.

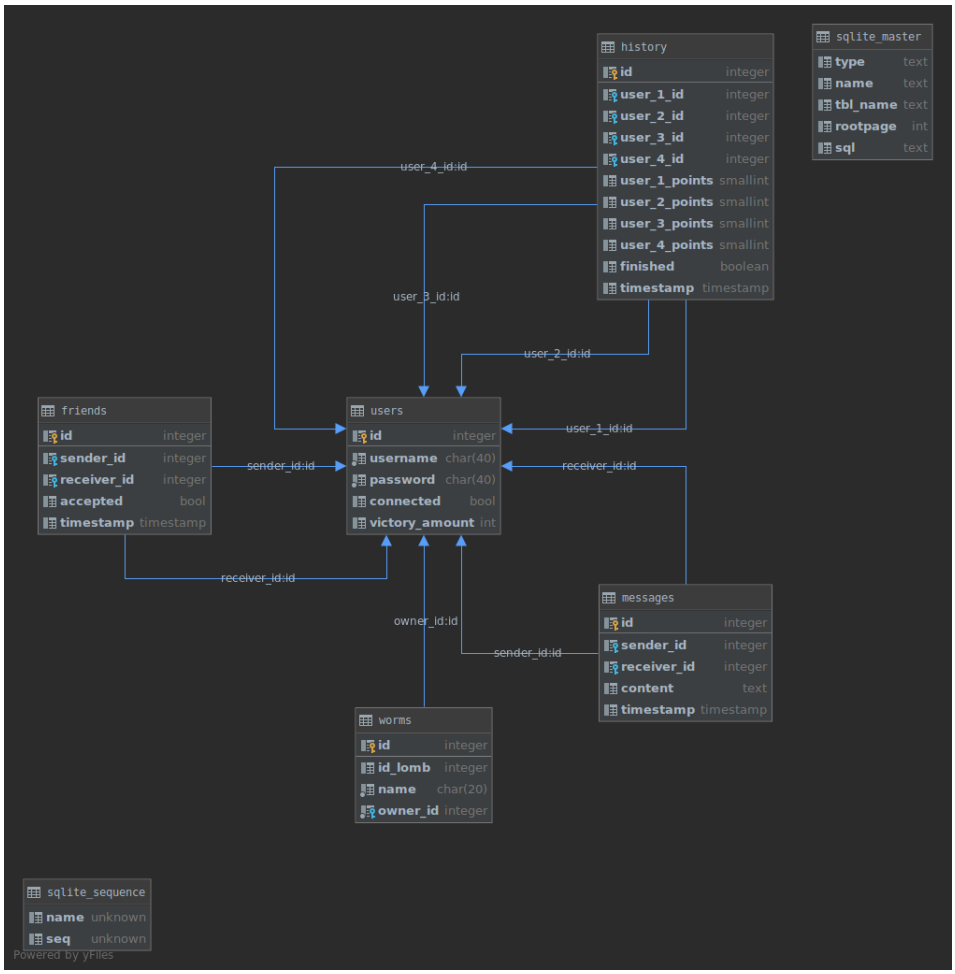

FIGURE 11 – Diagramme de la Base De Donnée

Diagrammes

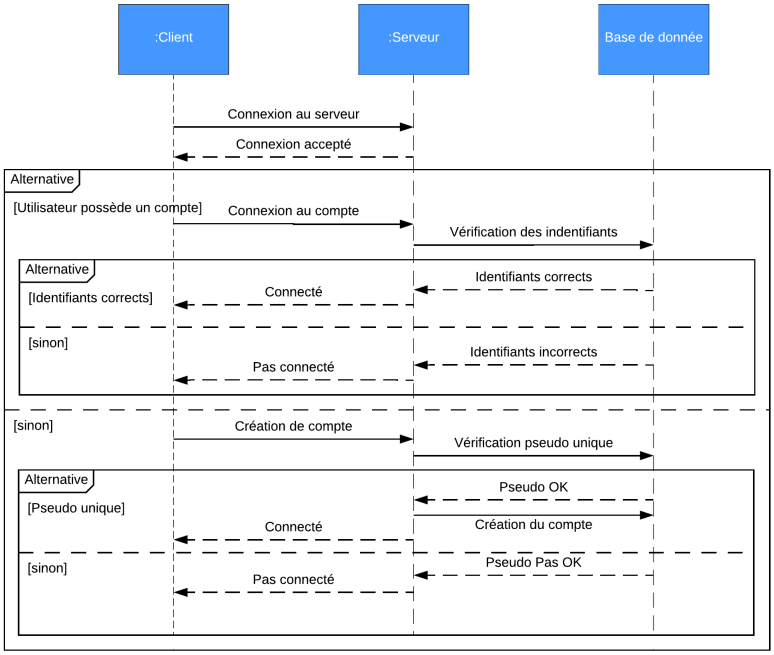

FIGURE 12 – Diagramme serveur client pour la connexion/inscription

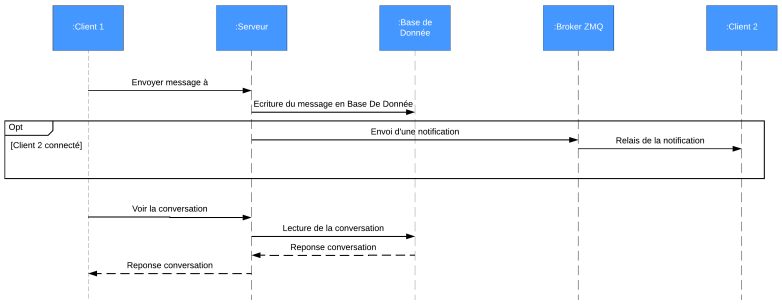

FIGURE 13 – Diagramme envoi de message