Quiddich live :
Software requirements document

Bruno Rocha Pereira Jérome Vial Cédric Strebelle
Romain Fontaine Tsotne Shonia Nikita Marchant

18 mars 2014

Table des matières

Chapitre 1

Introduction

1.1 But du projet

Le but du projet d'année en BA2 en sciences informatiques est le développement d'une application de jeu multijoueur en ligne. Il consiste en l'implémentation d'un jeu de Quidditch, sport tiré de la saga Harry Potter de J.K.Rowling.

Le Quidditch est un sport pratiqué par deux *équipes* de 7 *joueurs* évoluant à l'aide de leur balais sur un terrain de forme ovale. Le but de ces 2 *équipes* est de marquer le plus de points possible en marquant à l'aide du souaffle dans un des 3 buts adverses et d'attraper le vif d'or [1], ce qui signe la fin du match.

Notre application sera un jeu de gestion et stratégie en ligne, se jouant individuellement pendant la gestion et au tour par tour à 2 joueurs lors des matchs. L'*utilisateur* est le *manager* d'un *club* de Quidditch et son but est de faire progresser son *club* vers le succès. Pour ce faire, il devra gérer tous les aspects du *club*, tant au niveau sportif que financier. Il aura à sa disposition différentes installation pour entraîner ses *joueurs* ou gagner de l'argent. Il pourra améliorer les différentes installation de son *club* à l'aide d'argent.

Cet argent sera récolté lors des matchs et grâce aux revenus générés par les différents magasins du *club*. La saison de Quidditch se déroule sous la forme d'un championnat, durant lequel tous les *utilisateurs* inscrits s'affrontent entre eux au cours de matches. Les matches seront planifiés dans un calendrier qu'il faudra respecter sous peine de sanction. Un match se joue au tour part tour, mais les deux *utilisateurs* jouent simultanément, le *serveur* résolvant les conflits à la fin de chaque tour. Il y a donc une notion importante d'anticipation dans la maîtrise de ce jeu.

En ce qui concerne la gestion du *club*, l'*utilisateur* aura la possibilité de recruter de nouveaux *joueurs* de Quidditch pour renforcer son *équipe* via un système d'*enchères*, il pourra aussi développer différentes infrastructures et services comme son stade, son infirmerie, son terrain d'entraînement, son fanshop etc.

1. Pour plus d'informations, se référer au livre J. K. Rowling : "Le Quidditch à travers les âges", Gallimard, 2001.

1.2 Glossaire

client Programme que l'*utilisateur* lance pour se connecter au *serveur*. Il permet à l'*utilisateur* de jouer et peut être soit en ligne de commande, soit en mode graphique.

club Infrastructure contenant une *équipe* et des installations (stade, infirmerie,...).

enchère Système permettant l'achat de *joueurs*

joueur Personnage fictif du jeux pouvant faire partie d'une équipe et jouant au quiditch.

manager *Utilisateur* du programme, il possède un *club* et dois faire en sorte que celui-ci gagne de l'argent.

participant Nom donné à l'*utilisateur* lors d'un tournoi.

serveur Programme étant toujours en marche et connecté à internet et qui réceptionne les connections des *clients*. Il a les données de tous les *utilisateurs*.

utilisateur Personnage réel. C'est la personne qui joue au Quiditch manager. Elle est appelée en jeu *manager*

équipe Elle est composée de 7 *joueurs* (un gardien, un attrapeur, deux batteurs et trois poursuive).

1.3 Historique

0.1 (05/12/13) : Création du document - Equipe

0.2 (10/12/13) : Premier jet de chaque partie - Equipe

0.3 (11/12/13) : Création des diagrammes UML - Bruno, Tsotne

0.4 (11/12/13) : Amélioration des différents requirements - Nikita

0.5 (12/12/13) : Ajout du glossaire - Romain

0.6 (14/12/13) : Ajout du use case d'authentification et ajout des descriptions des use case - Tsotne

0.7 (15/12/13) : Finalisation des use case - Tsotne

0.8 (15/12/13) : Ajout de l'index et de la table des figures - Nikita

0.9 (19/12/13) : Corrections et ajout des diagrammes au document - Bruno, Tsotne

1.0 (20/12/13) : **Délivrable pour la phase 1 - Equipe**

1.1 (16/02/14) : Correction de fautes d'orthographes et remplissage de l'historique - Bruno

1.2 (18/02/14) : Ajout et correction de diagrammes de classe - Bruno

1.3 (20/02/14) : Modification du type de document et ajout des chapitres - Romain

1.4 (20/02/14) : Ajout du justificatif de choix de librairie graphique - Bruno

1.5 (21/02/14) : Ajout des diagrammes d'activité - Bruno

1.6 (21/02/14) : Finalisation de la justification de choix de librairie graphique - Bruno

1.7 (21/02/14) : Ajout des diagrammes de composants - Cédric

2.0 (21/02/14) : **Délivrable pour la partie 2 - Equipe**

2.1 (12/03/14) : Correction du diagramme de classe général - Bruno

2.2 (13/03/14) : Ajout du paragraphe "Idées de fonctionnalités" - Bruno

Chapitre 2

Besoins d'utilisation

2.1 Exigences fonctionnelles

2.1.1 Identification

2.1.2 Inscription (fig 2.1)

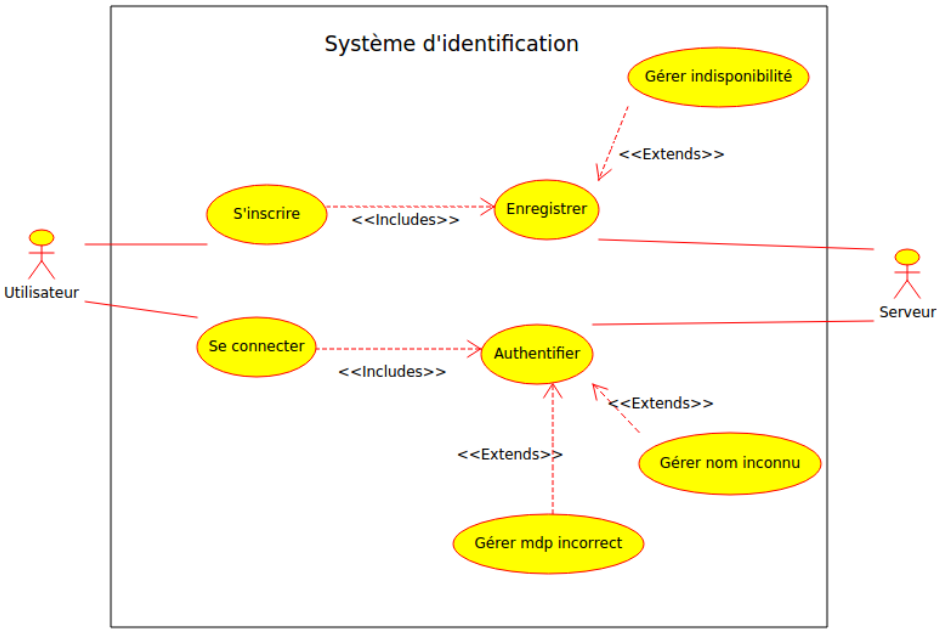


FIGURE 2.1 – Système d'identification

Cas général : L'*utilisateur* doit pouvoir s'inscrire (s'enregistrer) sur le *serveur* avec un nom d'*utilisateur* unique et un mot de passe.

Pré Condition : Néant.

Post Condition : L'*utilisateur* aura fait une demande d'inscription.

Cas Exceptionnel : L'inscription échouera et l'*utilisateur* sera invité à recommencer si le nom de compte est déjà enregistré sur le *serveur*.

2.1.3 Connection (fig 2.1)

Cas général : L'*utilisateur* doit pouvoir se connecter en s'authentifiant auprès du *serveur* avec son nom d'*utilisateur* et son mot de passe.

Pré Condition : Néant.

Post Condition : L'*utilisateur* aura fait une demande de connexion.

Cas Exceptionnel : L'authentification échouera et l'*utilisateur* sera invité à recommencer si le mot de passe ne correspond pas au nom d'*utilisateur*, ou cet *utilisateur* n'existe sur le *serveur*.

2.1.4 Gérer le *club* (en phase de management)

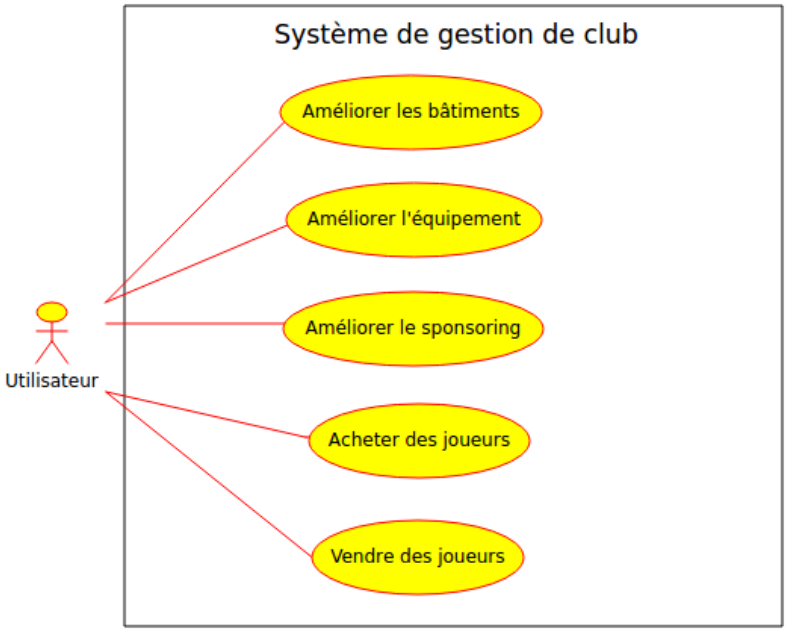


FIGURE 2.2 – Gestion d'un *club* pendant la phase de management

2.1.5 Améliorer les bâtiments

Cas général : L'*utilisateur* doit pouvoir améliorer ses divers bâtiments sur plusieurs niveaux.

Pré Condition : L'*utilisateur* doit être connecté.

Post Condition : Chaque bâtiment a sa ou ses spécificités, parmi ces trois-ci :

— Améliorer la rentrée d'argent.

— Améliorer l'entraînement (les caractéristiques) des *joueurs*.

— Apporter des soins plus efficaces aux *joueurs*.

Cas Exceptionnel : Une dépense n'est pas acceptée si le *club* de l'*utilisateur* ne dispose pas de suffisamment d'argent pour l'effectuer.

2.1.6 Améliorer l'équipement

Cas général : L'*utilisateur* doit pouvoir améliorer l'équipement de ses *joueurs* (bâtons, ...).

Pré Condition : L'*utilisateur* doit être connecté.

Post Condition : L'équipement (et donc les statistiques) des *joueurs* seront modifiées.

Cas Exceptionnel : Une dépense n'est pas acceptée si le *club* de l'*utilisateur* ne dispose pas de suffisamment d'argent pour l'effectuer.

2.1.7 Améliorer le sponsoring

Cas général : L'*utilisateur* doit pouvoir améliorer son sponsoring et les détails relatifs pour sa rentrée d'argent.

Pré Condition : L'*utilisateur* doit être connecté.

Post Condition : Le sponsoring, et donc une source de rentrée d'argent sera modifiée.

Cas Exceptionnel : Une dépense n'est pas acceptée si le *club* de l'*utilisateur* ne dispose pas de suffisamment d'argent pour l'effectuer.

2.1.8 Acheter des *joueurs*

Cas général : L'*utilisateur* doit pouvoir acheter des *joueurs*.

Pré Condition : L'*utilisateur* doit être connecté.

Post Condition : L'*équipe* de l'*utilisateur* sera modifiée.

Cas Exceptionnel : Une dépense n'est pas acceptée si le *club* de l'*utilisateur* ne dispose pas de suffisamment d'argent pour l'effectuer.

2.1.9 Vendre des *joueurs*

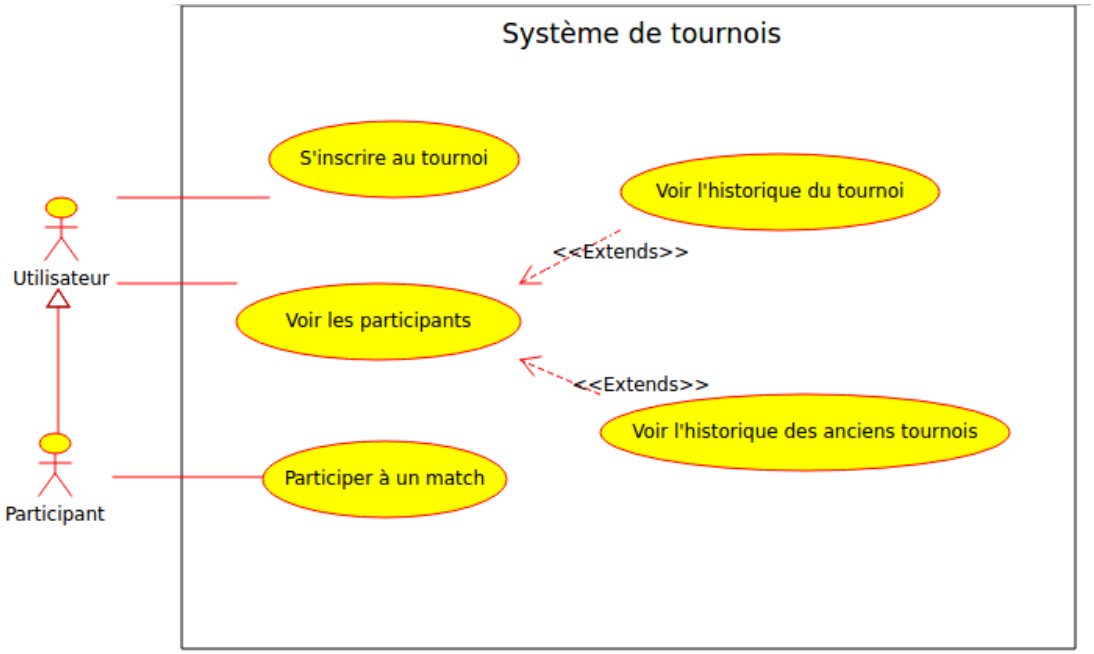
Cas général : L'*utilisateur* doit pouvoir vendre l'un (ou plusieurs) de ses *joueurs*.

Pré Condition : L'*utilisateur* doit être connecté.

Post Condition : L'*utilisateur* obtiendra de l'argent. Son *équipe* sera modifiée.

2.1.10 Jouer en tournoi

FIGURE 2.3 – Diagramme d'utilisation : gestion des tournois



2.1.11 S'inscrire en tournoi

Cas général : L'*utilisateur* doit pouvoir s'inscrire à un tournoi dont la date limite des inscriptions n'est pas encore passée. Ce qui lui permettra d'être sélectionné pour des matches et d'y affronter les autres *participants* inscrits à ce tournoi.

Pré Condition : L'*utilisateur* doit être connecté.

Post Condition : L'*utilisateur* sera inscrit pour le tournoi.

Cas Exceptionnel : Un *utilisateur* ne peut pas se réinscrire dans un tournoi auquel il est déjà inscrit.

2.1.12 Participer à un match du tournoi

Cas général : L'*utilisateur* doit pouvoir participer à un match en ligne du tournoi où il affrontera le *participant* adverse.

Pré Condition : L'*utilisateur* doit être connecté et sélectionné comme étant l'un des deux *participant* du match.

Post Condition : L'*utilisateur* sera en match contre le *participant* adverse.

Cas Exceptionnel : L'*utilisateur* sera considéré comme perdant automatiquement s'il a déclaré forfait.

2.1.13 Voir les informations relatives à un tournoi

2.1.14 Voir la liste des *participants* d'un tournoi

Cas général : L'*utilisateur* doit pouvoir voir la liste des *participants* d'un tournoi lancé ou terminé.

Pré Condition : L'*utilisateur* doit être connecté.

Post Condition : Néant.

2.1.15 Voir l'historique d'un tournoi

Cas général : L'*utilisateur* doit pouvoir voir l'historique d'un tournoi lancé ou terminé, c'est à dire l'arbre du tournois de ce dernier avec des détails concernant les *équipes*.

Pré Condition : L'*utilisateur* doit être connecté et ne doit pas être en plein match.

Post Condition : Néant.

2.1.16 Jouer à un match

2.1.17 Jouer un tour

Cas général : L'*utilisateur* doit pouvoir jouer un tour lors d'un match, c'est à dire déplacer ses *joueurs* et/ou effectuer des actions.

Pré Condition : L'*utilisateur* doit être entrain de jouer à un match.

Post Condition : L'*utilisateur* aura joué son mouvement.

Cas Exceptionnel : La perte de connexion avec un *utilisateur* maintenue, durant un certain timeout, entraînera son remplacement pour ce match par une IA fournie par le système. Le match continuera sans changement.

2.1.18 Déclarer forfait

Cas général : L'*utilisateur* peut déclarer forfait durant un match (il sera considéré comme perdant).

Pré Condition : L'*utilisateur* doit être entrain de jouer à un match.

Post Condition : L'*utilisateur* sera déclaré comme perdant.

2.2 Exigences non fonctionnelles

— La machine hébergeant le *client* ainsi que le *serveur* doivent être en mesure de communiquer en permanence via un réseau capable de transporter des paquets **TCP/IP**

— Le réseau décrit ci-dessus doit pourvoir une latence raisonnablement faible (c'est à dire de plus ou moins 200ms pour un aller retour)

— Les machines exécutant le *client* doivent être équipées d'un écran, d'un clavier et d'une souris et être capables d'afficher des images en mode graphique ainsi qu'au minimum 80 caractères de large en mode console. Elles devront aussi avoir au minimum disposer de 512Mb de mémoire vive ainsi que 500Mb d'espace disque.

2.3 Exigences de domaine

— Le jeu doit être multijoueur, les différents *utilisateurs* connectés sur un même *serveur* doivent pourvoir interagir entre eux.

— Le monde doit être persistent : il doit continuer d'évoluer, même en l'absence d'un ou plusieurs *utilisateurs*.

— Une *équipe* de Quiddich doit comporter 7 *joueurs* au maximum, sans compter les remplaçants.

— Un *joueur* blessé/mort ne peut être remplacé en plein match.

— Un match nécessite trois balles : Un souaffle, deux cognards et un vif d'or. Le terrain doit comporter deux buts, fait chacun de trois anneaux, et placé aux deux extrémités.

— Pour participer à un match, chaque *joueur* doit avoir au moins un balai.

— Le match ne prend fin que si le vif d'or est attrapé ou que l'une des deux *équipes* abandonne.

Chapitre 3

Besoin du système

3.1 Exigences fonctionnelles

3.1.1 Identification

3.1.2 Enregistrer une inscription

Cas général : Le système doit être capable d'enregistrer un nom de compte unique associé à un mot de passe dans un fichier, ainsi que de vérifier que le nom de compte fourni est unique.

Pré Condition : Un *utilisateur* effectuant une demande d'inscription.

Post Condition : Le nom de compte sera enregistré sur le *serveur*. L'*utilisateur* sera donc inscrit.

Cas Exceptionnel : L'enregistrement échouera si le nom de compte est déjà enregistré sur le fichier.

3.1.3 Authentifier

Cas général : Le système doit être capable d'authentifier un *utilisateur* demandant de se connecter en vérifiant que le nom de compte fourni lors de la connexion est présente dans le fichier du *serveur* et est bien associé au mot de passe fourni.

Pré Condition : Un *utilisateur* effectuant une demande de connexion.

Post Condition : L'*utilisateur* sera connecté.

Cas Exceptionnel : L'authentification échouera si le nom de compte fourni n'est pas enregistré sur le *serveur*, ou s'il est associé à un autre mot de passe que le mot de passe fourni.

3.1.4 Interface

3.1.5 Fournir une interface

Cas général : Le système doit fournir à l'*utilisateur* une interface jeu simple, complète et interactive, graphique et en console pour les deux phases de jeu.

Pré Condition : Néant.

Post Condition : Néant.

3.1.6 Représenter phase management

Cas général : Le système doit fournir à l'*utilisateur* une représentation du *club* (phase management), comprenant une vue d'ensemble sur son argent, ses rentrées, ses *joueurs*, ses infrastructures et les améliorations possibles.

Pré Condition : Néant.

Post Condition : Néant.

3.1.7 Représenter phase match

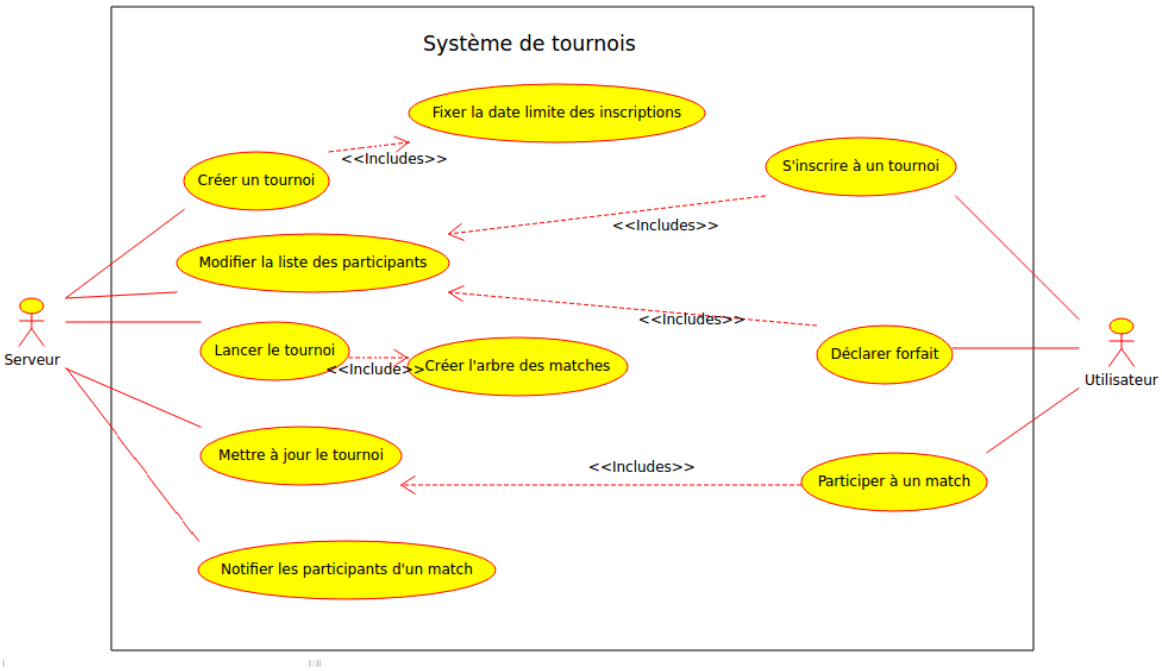
Cas général : Le système doit fournir à l'*utilisateur* une représentation du terrain ovale (phase match), sous forme de cases hexagonales.

Pré Condition : Néant.

Post Condition : Néant.

3.1.8 Gestion de tournois

FIGURE 3.1 – Gestion de tournois.



3.1.9 Créer un tournoi

Cas général : Le système doit créer les tournois avec un certain intervalle de temps, où les *utilisateurs* pourront s'inscrire et devront s'affronter durant des matches organisés dans les stades des *clubs*.

Pré Condition : Néant.

Post Condition : Un tournoi sera crée et mit en phase d'attente. Durant sa création, il fixera une date limite pour les inscriptions.

3.1.10 Modifier la liste des *participants*

Cas général : Le système doit gérer une liste des *participants*, indiquant quel *utilisateur* participe ainsi que son état (éliminé, en course). Le système doit être capable de le modifier la liste durant les inscriptions et après que le tournoi soit lancé.

Pré Condition : Un tournoi en attente ou lancé.

Post Condition : La liste des *participants* d'un tournoi sera modifiée.

3.1.11 Lancer le tournoi

Cas général : Lorsque la date limite d'une phase d'inscription tournoi en attente est atteinte, le système doit générer l'arbre du tournois et lancer le dit tournoi. L'arbre du tournois contient également la date et l'heure de chaque match.

Pré Condition : Être au moment de lancement du tournoi.

Post Condition : Le tournoi sera lancé.

3.1.12 Notifier les *utilisateurs*

Cas général : Le système se doit de notifier les deux *utilisateurs* concerné par un match lorsque la date et l'heure de celui-ci est atteint. Si un *utilisateur* ne répond pas, après un certain timeout, il est considéré comme perdant.

Pré Condition : Néant.

Post Condition : Les *utilisateurs* seront notifiés pour participer au match.

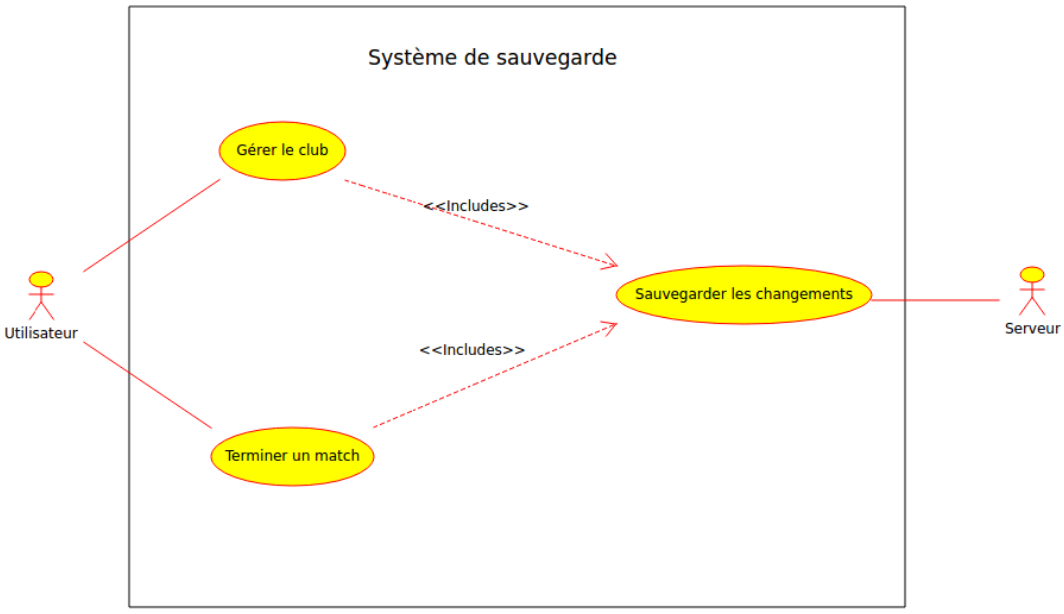
3.1.13 Mettre à jour un tournoi

Cas général : Le système doit mettre à jour l'arbre du tournoi après chaque match en y indiquant les vainqueurs et les éliminés. Les vainqueurs pourront s'affronter aux matches suivants et les perdants seront éliminés, jusqu'à ce qu'il ne reste qu'un seul *participant* qui sera élu grand vainqueur.

Pré Condition : Une fin de match entre deux *participants*.

Post Condition : L'arbre de tournoi sera mis à jour.

FIGURE 3.2 – Sauvegarde



3.1.14 Divers

3.1.15 Sauvegarder

Cas général : Le système doit sauvegarder tout changement effectué au *club* durant la phase management ainsi qu'après une partie.

Pré Condition : Une modification du *club* en phase management, ou une fin de match.

Post Condition : Les changements apportés au *clubs* et au *serveur* seront sauvegardés.

Cas Exceptionnel : Aucune sauvegarde ne sera effectuée par le module de sauvegarde si le *serveur* plante.

3.2 Exigences non fonctionnelles

— Le *client* et le *serveur* doivent être écrits en **C++** et seront compilés à l'aide de **gcc 4.8**

— Le *client* et le *serveur* doivent être portables et pouvoir fonctionner sur un système **UNIX** et une architecture x86

— La machine exécutant le *serveur* doit être capable de gérer une connexion ouverte constamment vers chaque *client* ainsi que de stocker l'entièreté

des données du jeu en mémoire disque ainsi qu'une grande majorité en mémoire vive.

3.3 Design et fonctionnement

3.3.1 Diagrammes de classe

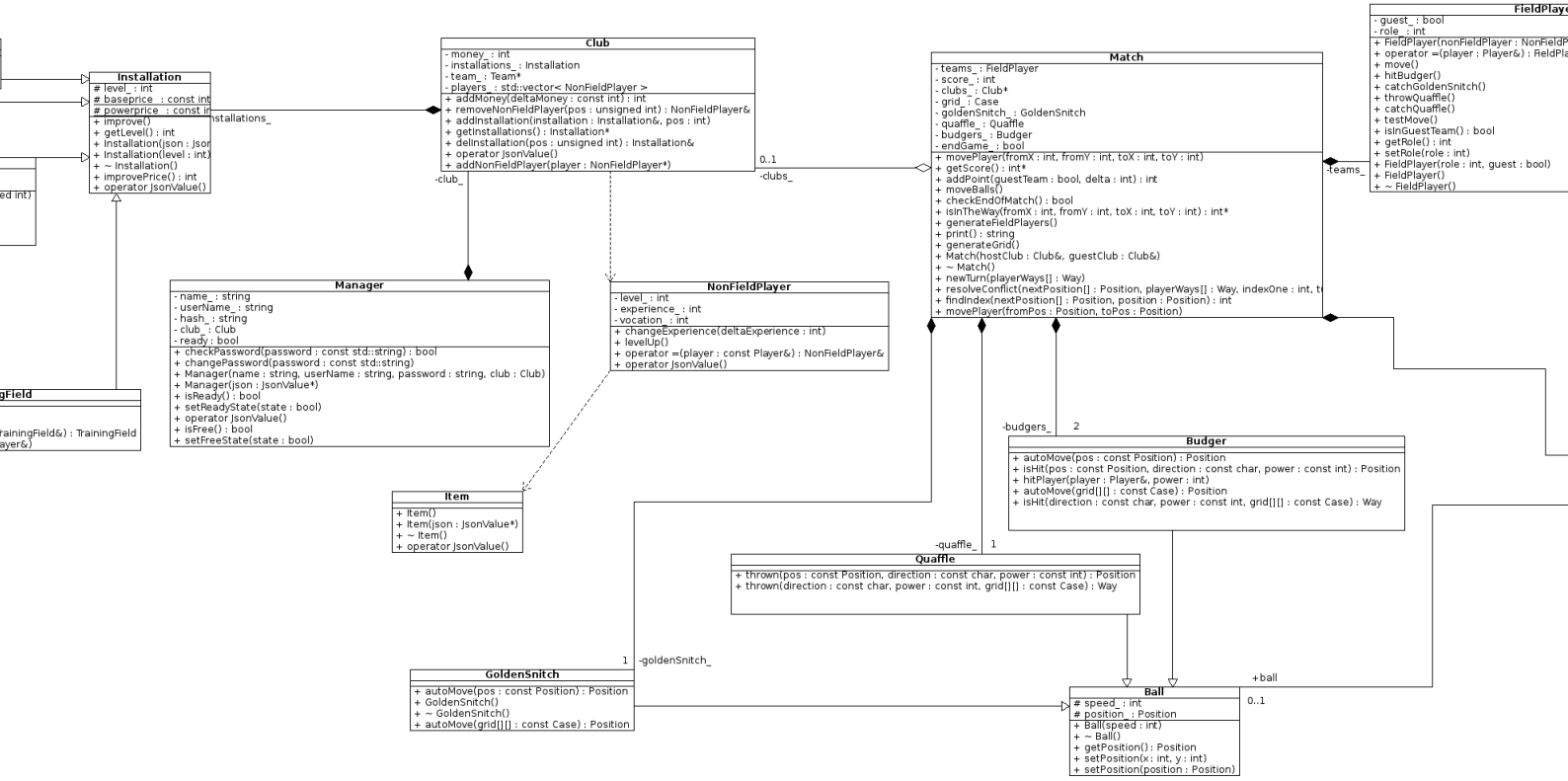


FIGURE 3.3 – Diagramme de classe représentant toutes les structures de données

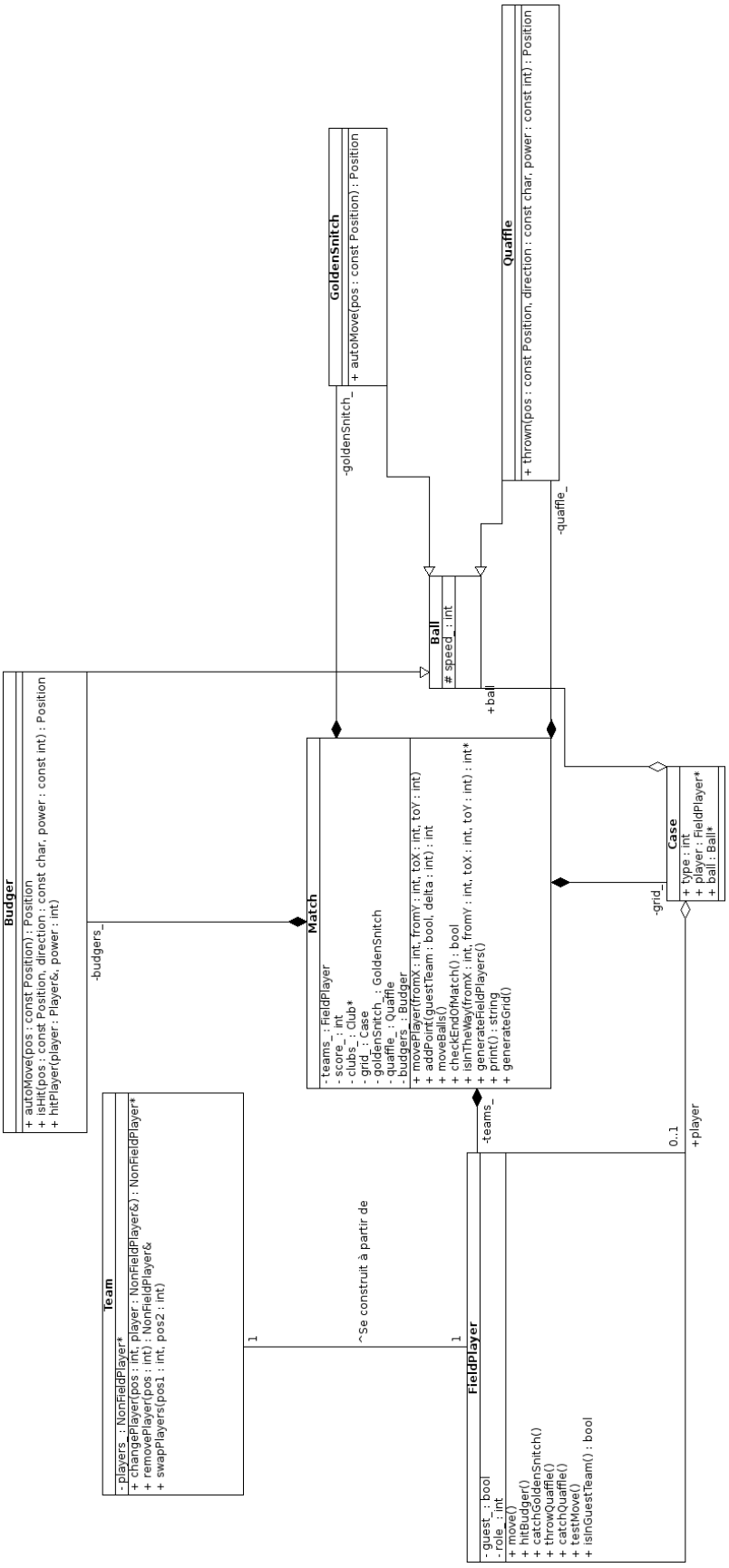


FIGURE 3.4 – Diagramme de classe représentant les structures de données intervenant pour un match

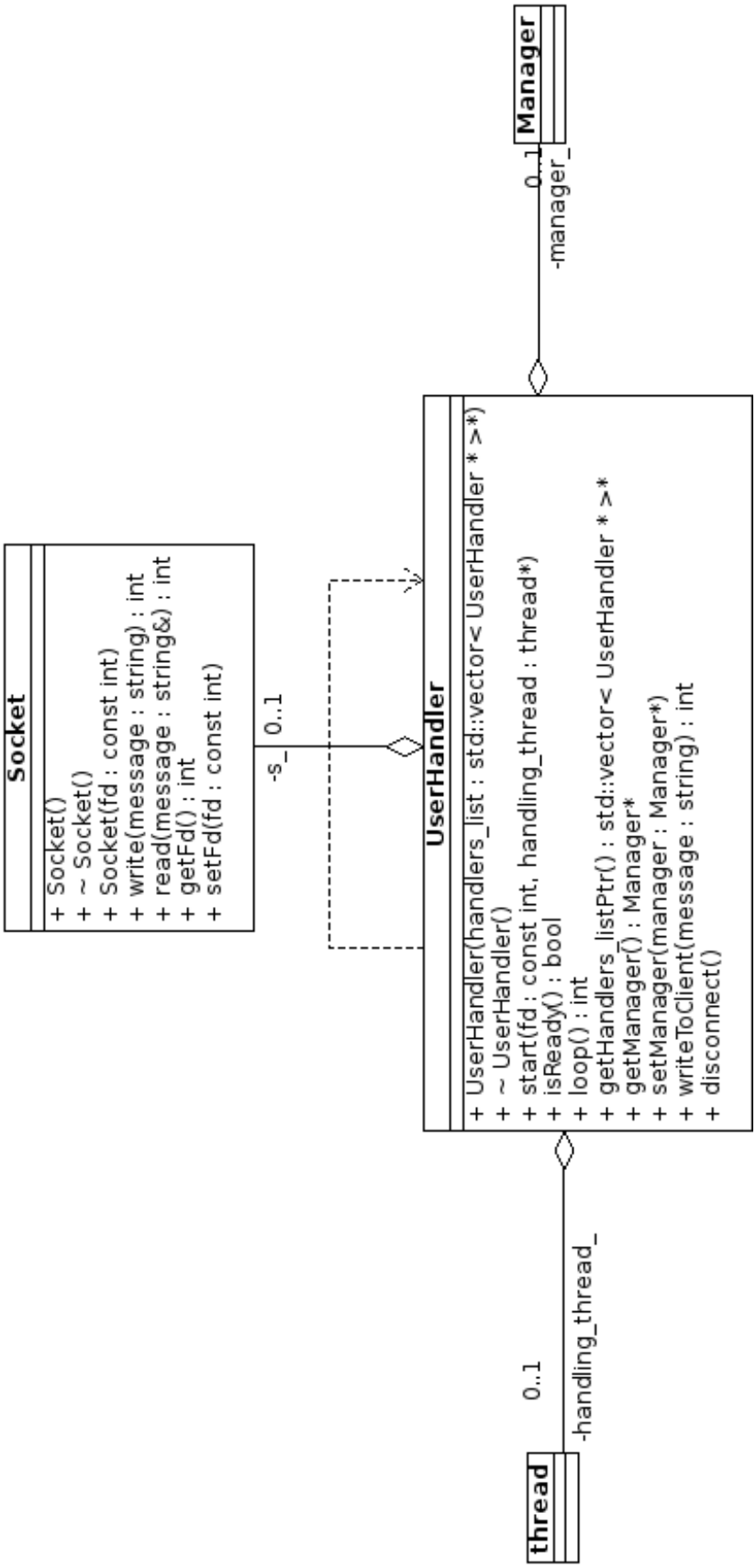


FIGURE 3.5 – Diagramme de classe représentant les structures de données intervenant pour la partie serveur

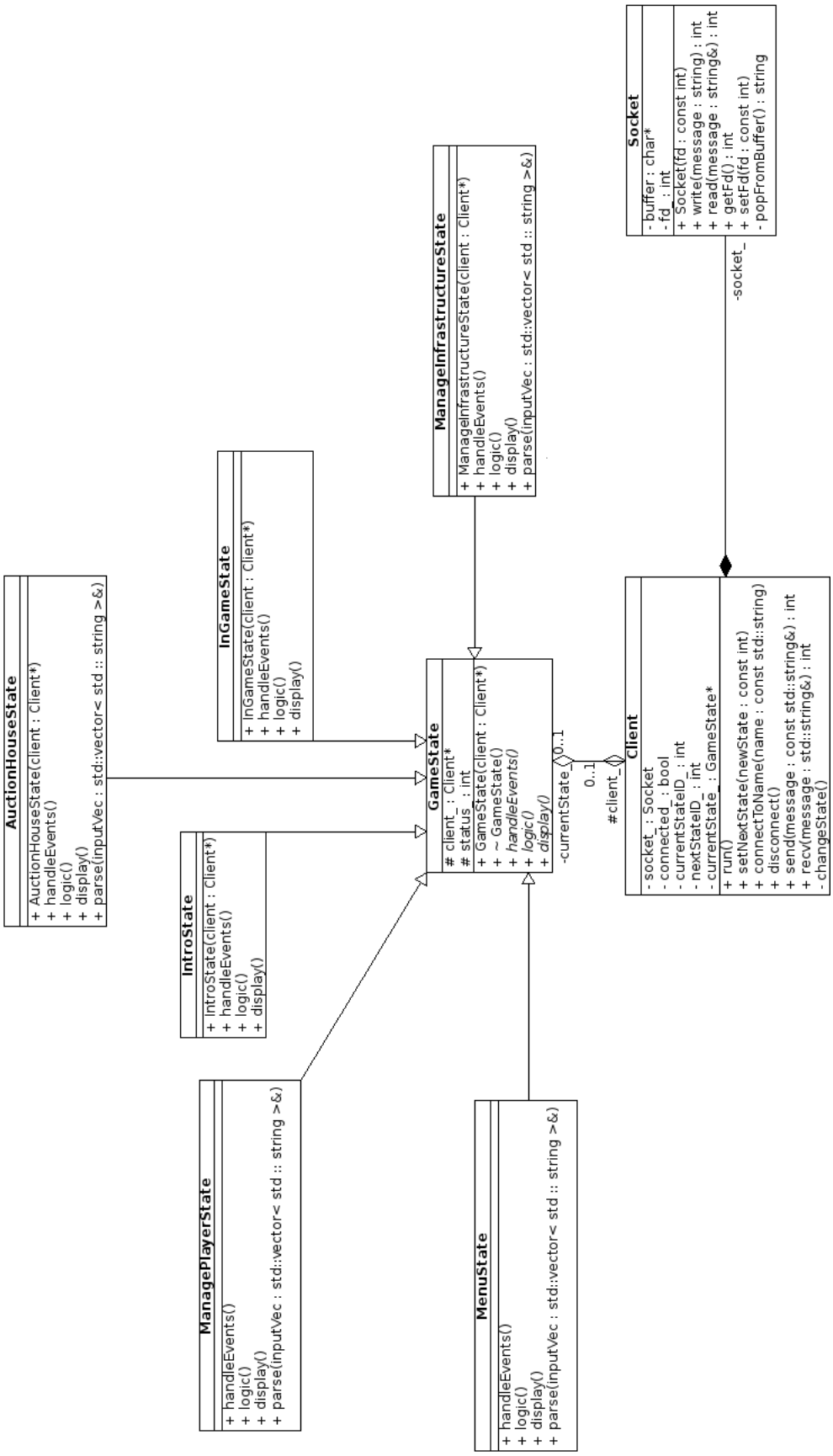


FIGURE 3.6 – Diagramme de classe représentant les structures de données intervenant pour la partie client

3.3.2 Diagrammes d'activité

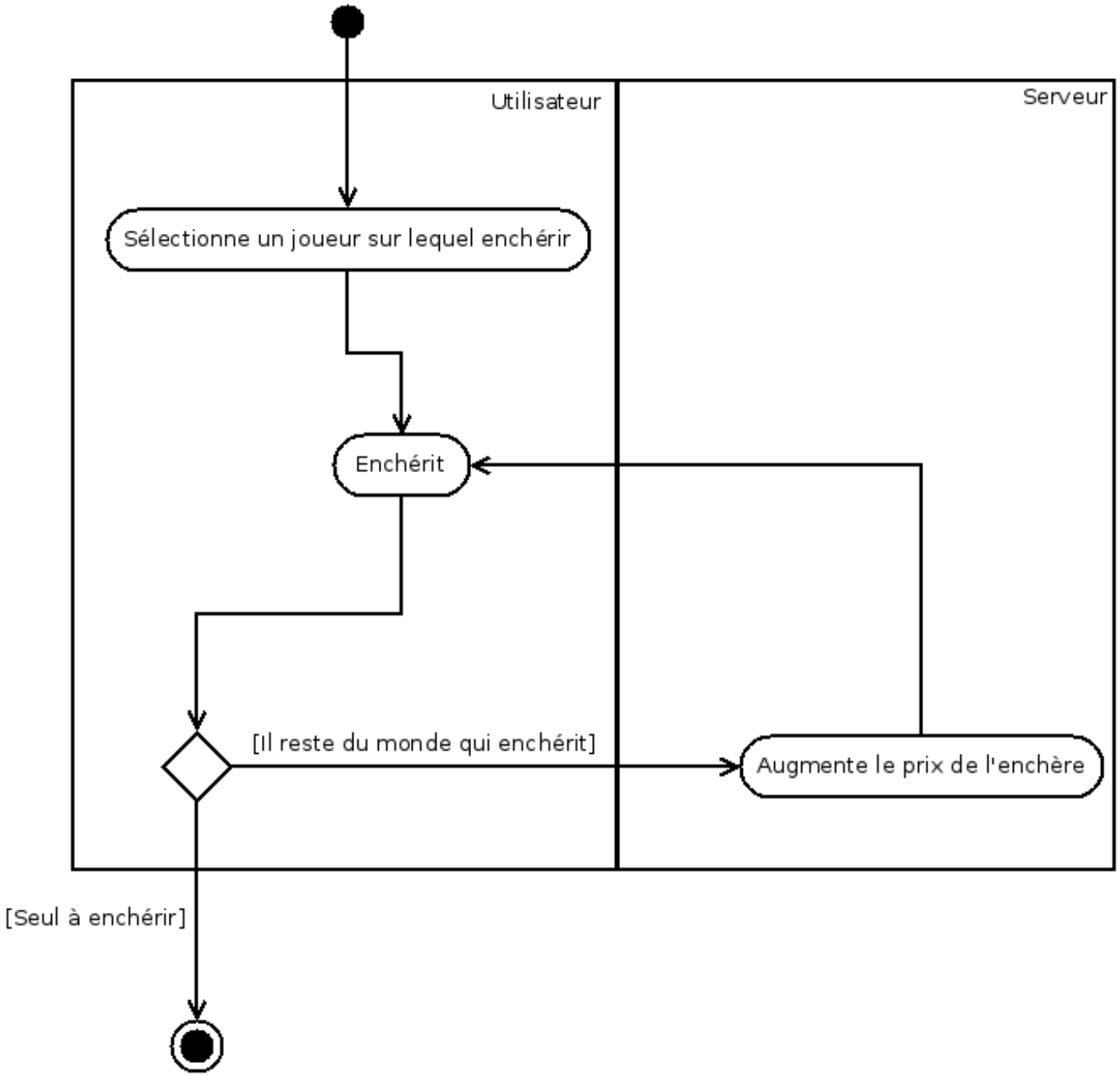


FIGURE 3.7 – Diagramme d'activité représentant le déroulement d'une enchère

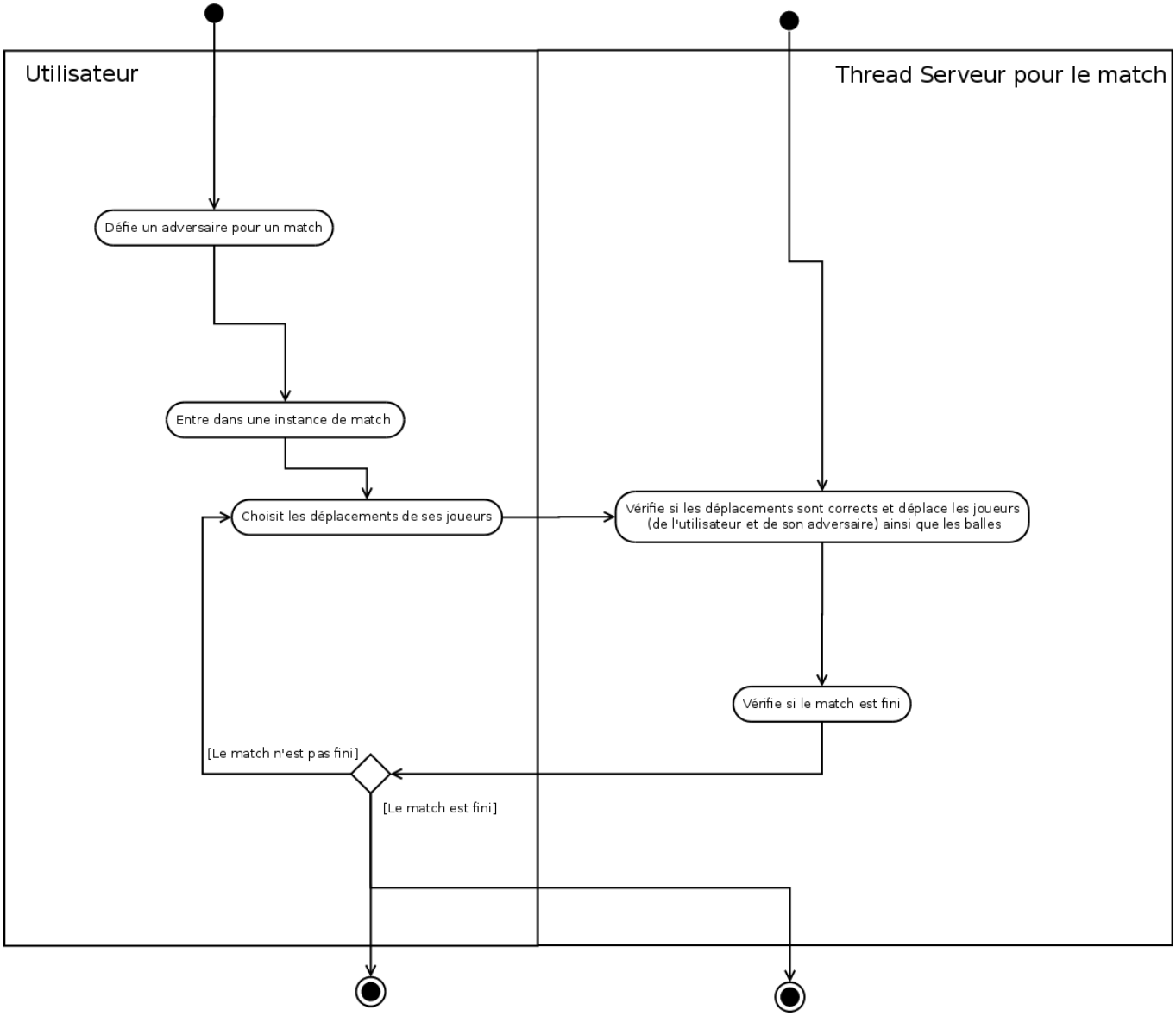


FIGURE 3.8 – Diagramme d'activité représentant le déroulement d'un match

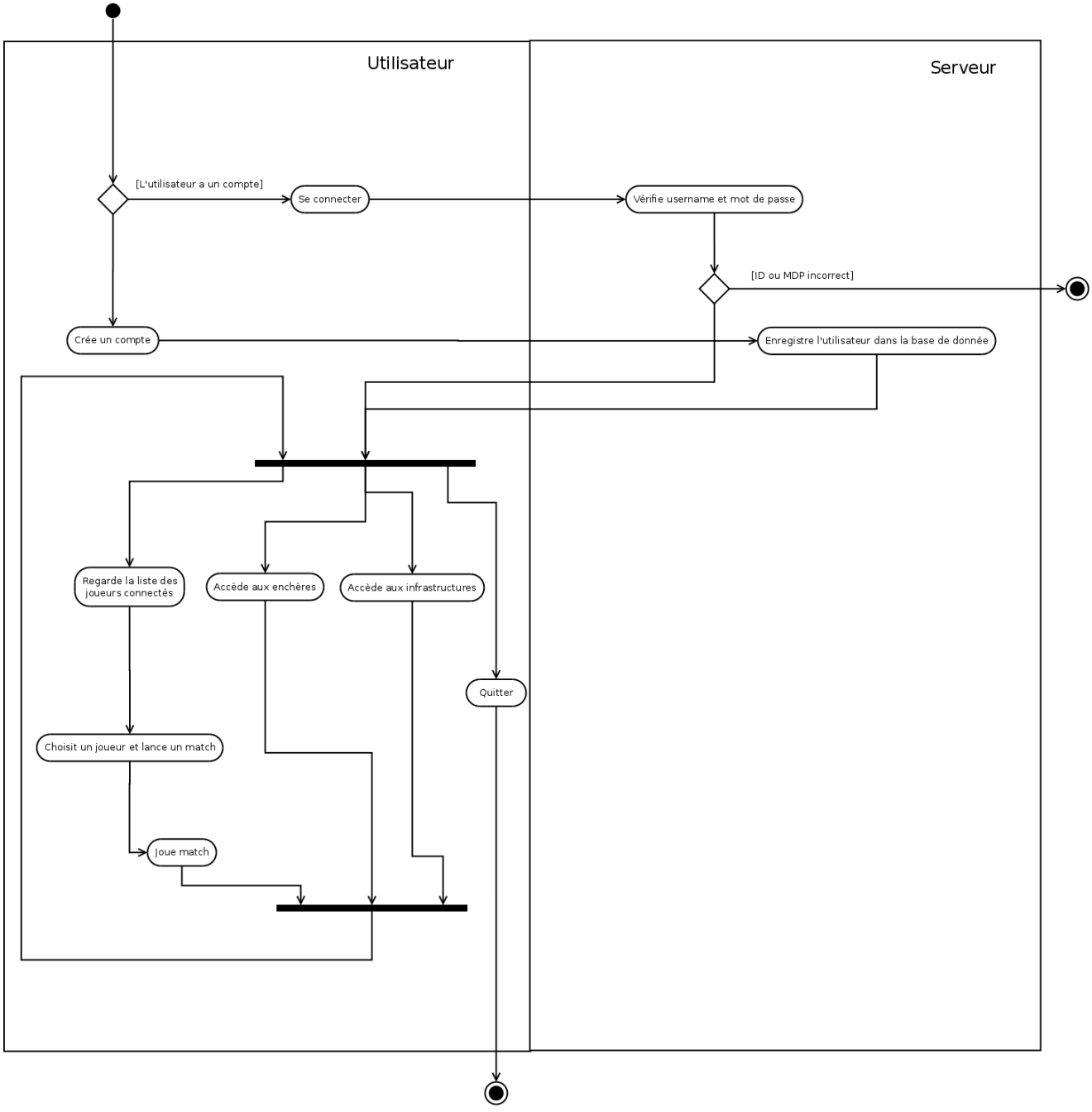


FIGURE 3.9 – Diagramme d'activité représentant la procédure normale d'une partie de jeu

3.3.3 Diagrammes de composants

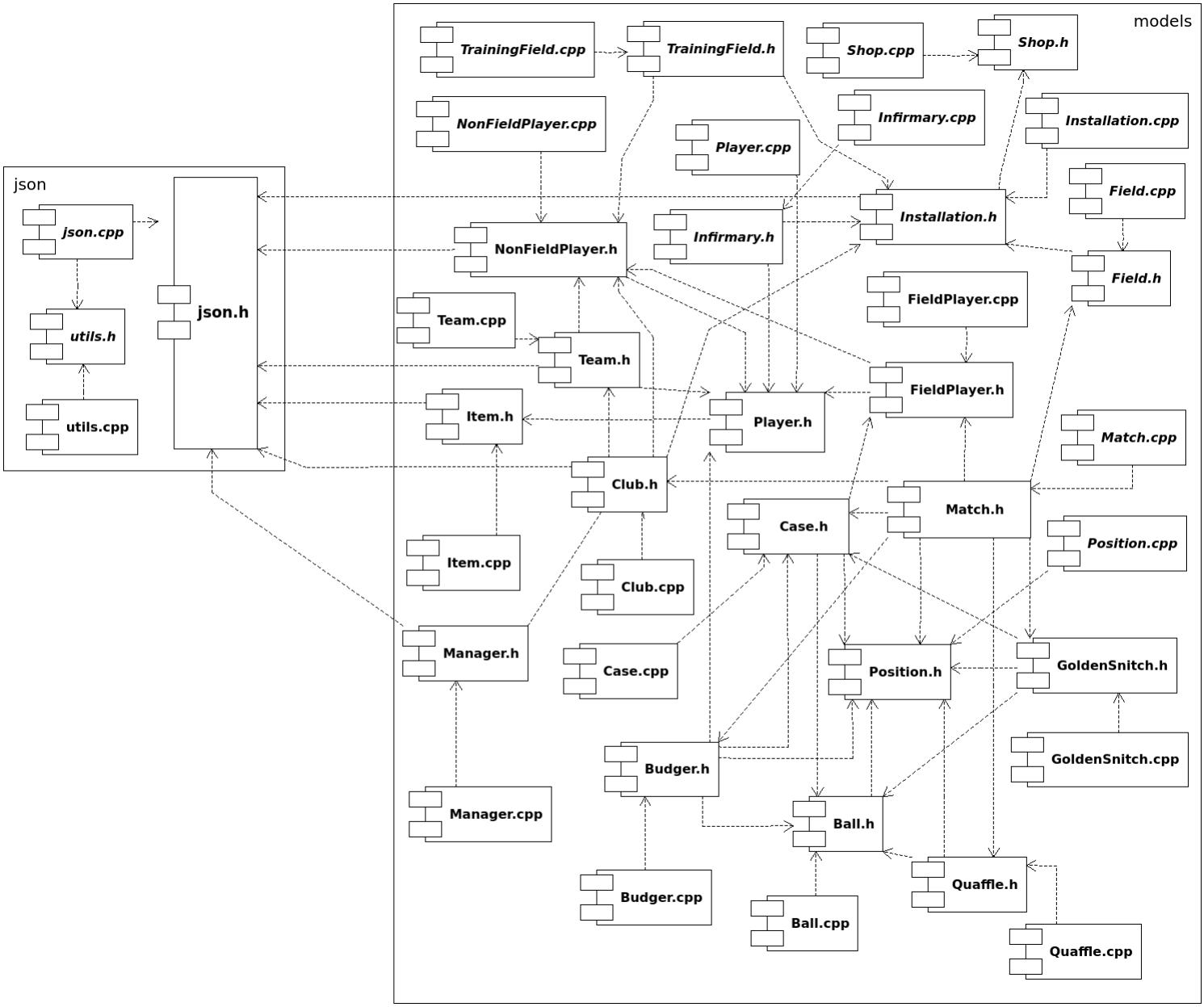


FIGURE 3.10 – Diagramme de composants représentant les dépendances au sein du répertoire lib/json et models

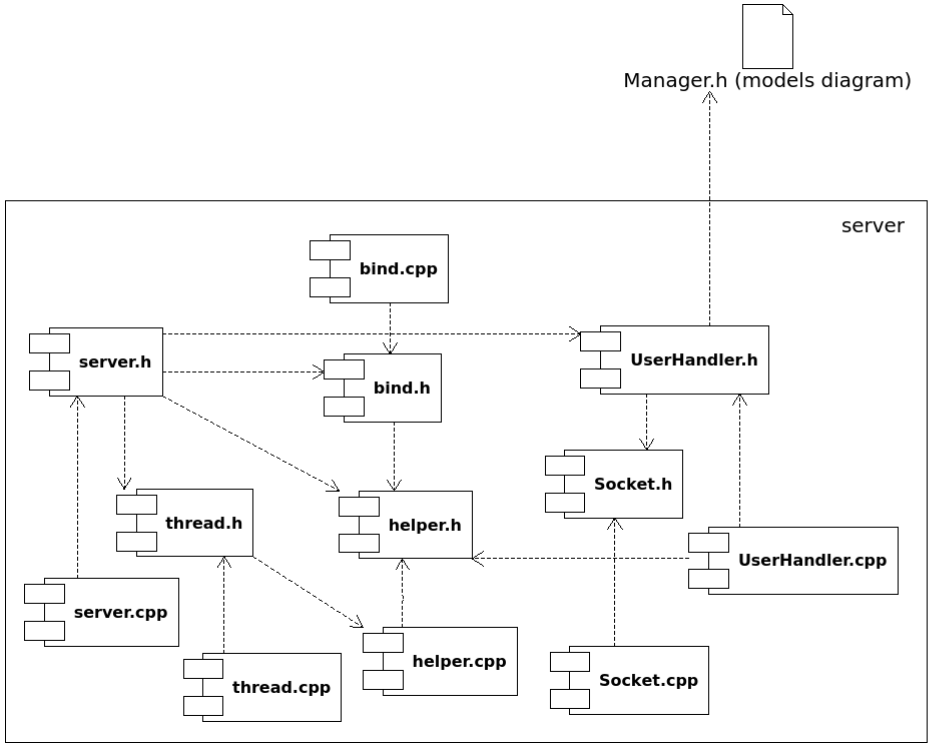


FIGURE 3.11 – Diagramme de composants représentant les dépendances au sein du répertoire server

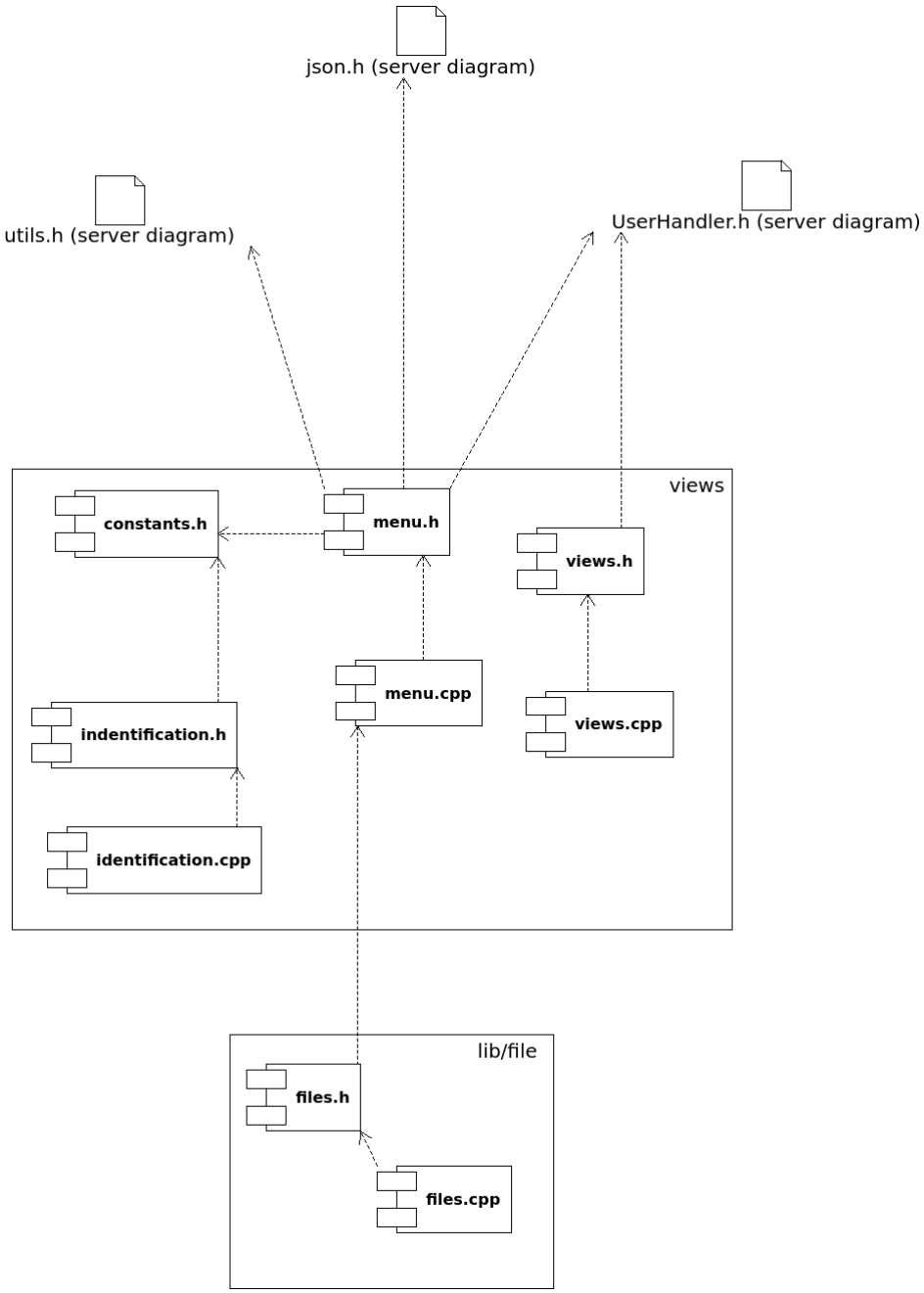


FIGURE 3.12 – Diagramme de composants représentant les dépendances au sein du répertoire lib/file et views

Chapitre 4

Justification de la bibliothèque graphique

4.1 Qt

4.1.1 Avantages

— Qt est une librairie complête et possède une documentation énorme. De ce fait, il sera facile de trouver réponse à nos questions.

— Qt est une librairie utilisée depuis 1995 et le projet est toujours actif et maintenu par une grande société (Nokia), on peut donc la considérer comme stable.

— Qt est écrit en C++ et donc nativement adapté pour ce langage.

— Qt est OpenSource (sous license GNU LGPL).

— Qt permet d'utiliser le framework QgraphicsView, facile d'utilisation et complet, permettant de dessiner aisément ce que l'on veut.

— On peut utiliser Qt Creator pour cette librairie graphique, IDE fournissant une autocomplétion pour tous les modules de Qt et possédant une aide disponible intrinsèque.

4.1.2 Inconvénients

— Lourd

— Possède des packages qui ne nous serviront pas, étant donné que nous ne n'utiliserons que la partie graphique de Qt

4.2 GTK+

4.2.1 Avantages

— Plus léger

4.2.2 Inconvénients

— Documentation non complête

— Pas d'IDE fournie par l'équipe en charge du projet GTK+

4.3 Conclusion

Nous utiliserons donc Qt, de par la présence et la simplicité d'utilisation de son IDE *Qt Creator* et par la richesse de la documentation qui l'accompagne. La stabilité de cette API fut également un élément influencant notre choix.

Chapitre 5

Idées d'ajout de fonctionnalités

5.1 A finir

— Les *enchères*
— Un système d'admin
— Gestion d'équipes
— Points d'actions
— Sponsors

5.2 Fonctionnalités supplémentaires

— Équipes nationales
— Événements aléatoires
— Boîte à message
— Statistiques

Chapitre 6

Index

Table des figures