Quidditch Manager 2014

Software Requirements

Document

Projet d'année BA2 INFO - ULB

Groupe 4 : Manon Legrand, Hélène Plisnier, Audry Delestree, David Bertha
INFOF209 - Projet d'informatique - Quidditch Manager 2014
Université Libre de Bruxelles
Version 3.0

Table des matières

1 Introduction

1.1 But du projet

Le Quidditch manager est un jeu multi-joueurs de gestion et de stratégie, se jouant en réseau. Un manager (l'utilisateur) gère son équipe de joueurs, qu'il peut entrainer au travers des bâtiments construits sur la parcelle qui lui est attribuée. Certains de ces bâtiments permettent aussi de faire des achats et des ventes de balais ou de joueurs. Le but pour le manager est de développer son club aussi bien au niveau commercial que sportif. À travers la construction et l'évolution de ses bâtiments, le manager ouvre et augmente des possibilités de gestion du club, notamment la possibilité d'améliorer les caractéristiques de ses joueurs, ces caractéristiques étant déterminantes dans le déroulement d'un match. Les différents managers sont périodiquement invités à s'affronter par ces matchs s'inscrivant dans un championnat, étalonné dans le temps.

Ces matchs se déroulent dans un stade propre à un des managers et apportent une entrée d'argent dûe notamment à la victoire d'une équipe mais aussi au nombre de tickets vendus. Au cours du match, les managers en présence choisissent à chaque tour de jeu des déplacements pour leurs différents joueurs. Il existe quatre fonction possible pour un joueur lors d'un match. Un joueur peut être un attrapeur, un batteur, un gardien ou un poursuiveur. Chacun de ces rôles implique des actions possibles différentes au cours du match. Mais il est possible à chacun de se déplacer sur le terrain, celui-ci étant représenté par des hexagones (il y a donc 6 directions de déplacement possibles à partir d'une case). De plus, les joueurs se partagent le terrain avec les différentes balles du jeu, qu'ils devront manipuler selon les règles pour emmener leur équipe vers la victoire, celle-ci étant déterminée lorsque le vif d'Or est attrapé ou par abandon.

1.2 Historique du document

Numéro de version	Auteur	Date de la modification	Description des changements
0.1	David	13/12/13	Squelette du code LaTeX
0.2	Hélène	19/12/13	Première ébauche de la section 2.1
0.3	Manon	20/12/13	Section 2.1 finale
0.4	David	20/12/13	Sections 1, 2.2 et 2.3
0.5	Manon	20/12/13	Section 3.1
0.5	David	20/12/13	Section 3.2
0.6	Audry	20/12/13	Section 3.3
0.7	David	20/12/13	Glossaire et Index
1.0	Manon	20/12/13	Relecture et révision
1.1	Hélène	07/02/14	1ère correction
1.2	Hélène	20/02/14	Communication réseau et Qt
1.3	Audry	21/02/14	Nouveaux diagrammes de classes
1.4	Manon	21/02/14	Commentaires par rapport aux Use Case
1.5	Manon	21/02/14	Explications des diagrammes de classes
1.6	Audry	21/02/14	Nouveaux diagrammes de classes (2)
1.7	Manon	21/02/14	Modif de certains commentaires de diag
2.0	Manon	21/02/14	Jeu sur terminal et Echange de messages
2.1	David	11/03/14	Précisions de l'architecture réseau
2.2	Manon	06/03/14	Titre + Section 2.1
2.3	Manon	12/03/14	Amélioration diags de classes
2.4	David	12/03/14	Diagramme de composants
2.5	David	12/03/14	Précisions sur le déroulement d'un match
2.6	David	13/03/14	Diagrammes pour la GUI
2.7	Hélène	14/03/14	Terminal graphique avec diagrammes détaillés

1.3 Glossaire

2 Besoins de l'utilisateur

2.1 Exigences fonctionnelles

Les cas d'utilisation présentés ci-dessous décrivent les différentes actions proposées au client par le programme.

2.1.1 Cas d'utilisation 1 : Démarrer le jeu

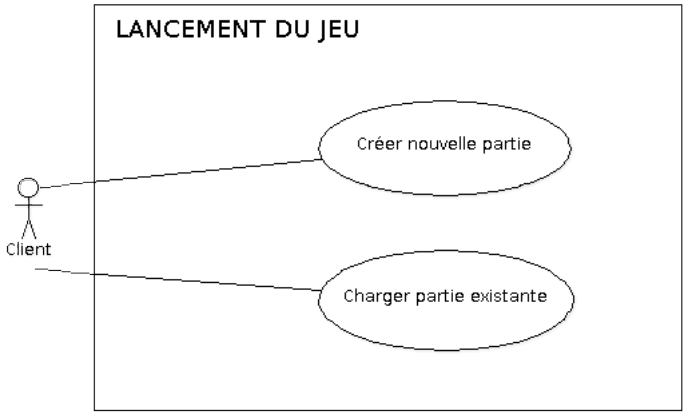

FIGURE 2.1: Cas d'utilisation 1 : Démarrer le jeu

L'utilisateur démarre le jeu. Soit il a déjà commencé une partie ; dans ce cas, l'utilisateur s'identifie et reprend sa partie là où il l'avait laissée.
Soit il ne possède pas encore de partie ; le programme lui demande alors quelques informations d'identification. Il se voit ensuite attribuer une équipe de 7 joueurs médiocres et une parcelle. Au commencement, ce dernier est presque désert, composé uniquement d'un stade de Quidditch et d'emplacements vides prévus pour accueillir les bâtiments.

Créer nouvelle partie

- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Le client désire se créer une nouvelle partie.
- **Post-conditions** : Le client possède une partie.
- **Cas général** : Le client veut créer une nouvelle partie. Il devra alors rentrer toute une série d'informations concernant cette partie nécessaire à l'identification de cette

partie et au bon fonctionnement du jeu (nom du manager, nom de l'équipe, mot de passe, etc.)
- **Cas exceptionnels** :
  - Un manager avec le nom indiqué par le client existe déjà. Le programme le signale au client qui doit de nouveau choisir ce qu'il veut faire.

Charger partie existante

- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Il faut qu'il existe déjà une partie à laquelle créée par ce client.
- **Post-conditions** : Le client a accès à sa partie.
- **Cas général** : Le client souhaite continuer une partie déjà entamée et rentre le nom de la partie ainsi que le mot de passe pour récupérer les informations concernant cette partie.
- **Cas exceptionnels** :
  - L'identification de la partie a échoué (mauvais nom ou mot de passe). Le programme le signale au client qui doit de nouveau choisir ce qu'il veut faire.

2.1.2 Cas d'utilisation 2 : Gestion de l'équipe

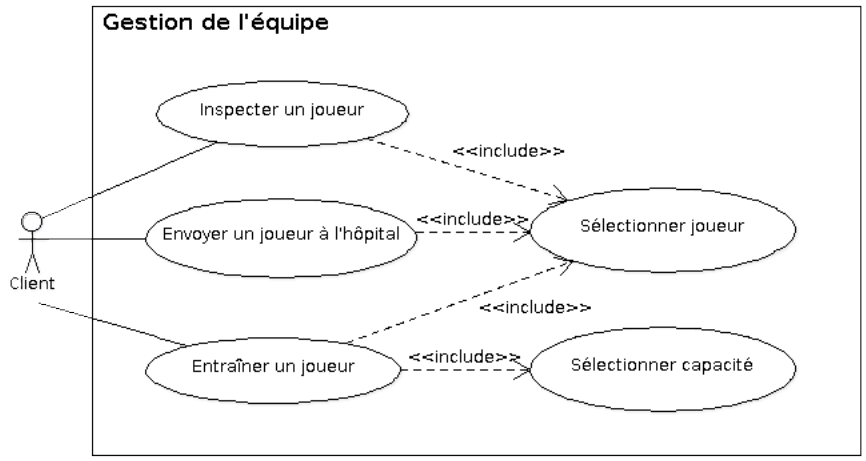

FIGURE 2.2: Cas d'utilisation 2 : Gestion de l'équipe

Une fois que l'utilisateur a rejoint sa partie ou en a créé une nouvelle, il peut gérer son équipe. Le manager peut alors au choix se renseigner sur ses joueurs, et donc voir les capacitiés de ce dernier, l'avancement de ses entraînements, etc. Il peut également envoyer un joueur au centre d'entraînement ou à l'hôpital.

Inspecter un joueur

- **Relations avec d'autres cas d'utilisation** : Inclut Sélectionner joueur.
- **Pré-conditions** : Néant.
- **Post-conditions** : Une fiche d'identitédu joueur est affichée.
- **Cas général** : Le client/manager souhaite se renseigner sur un de ses joueurs. Il sélectionne le joueur concerné pour avoir les informations sur ce dernier. Ces informations sont : les capacités, l'avancement des entraînements par capacité, une indication quant au blocage (entraînement/hôpital/enchères) éventuel du joueur, les informations sur le balai du joueur et une estimation de la valeur du joueur.
- **Cas exceptionnels** : Néant.

Envoyer un joueur à l'hôpital

- **Relations avec d'autres cas d'utilisation** : Inclut Sélectionner joueur.
- **Pré-conditions** : Le joueur sélectionné n'est pas bloqué par un entraînement, un séjour à l'hôpital ou une enchère et le client possède suffisamment de points d'action pour envoyer son joueur à l'hôpital.
- **Post-conditions** : Le joueur est soigné mais bloqué pour une durée qui dépend du niveau de l'hôpital.
- **Cas général** : Le client souhaite soigner un joueur qui a été blessé au cours d'un match et l'envoie donc à l'hôpital pour que lui soient procurés les soins requis.
- **Cas exceptionnels** : Néant.

Entraîner un joueur

- **Relations avec d'autres cas d'utilisation** : Inclut Sélectionner joueur et Sélectionner capacité.
- **Pré-conditions** : Le joueur sélectionné n'est pas bloqué par un entraînement, un séjour à l'hôpital ou une enchère et le client possède suffisamment de points d'action pour que son joueur commence un entraînement.
- **Post-conditions** : L'avancement de l'entraînement de la capacité concernée du joueur choisi est incrémenté, si l'avancement devient complet, c'est-à-dire si le nombre de sessions d'entraînement requis pour passer au niveau supérieur est atteint, la capacité est incrémentée et l'avancement est réinitialisé.
- **Cas général** : Le client souhaite entraîner un joueur pour le faire progresser dans une capacité.
- **Cas exceptionnels** : Néant.

Sélectionner joueur

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Inspecter joueur, Envoyer un joueur à l'hôpital et Entraîner joueur.
- **Pré-conditions** : Néant.
- **Post-conditions** : Le joueur est sélectionné.

- **Cas général** : Le client choisit le joueur qui l'intéresse pour l'inspection, l'envoi à l'hôpital ou l'entraînement.
- **Cas exceptionnels** : Néant.

Sélectionner capacité

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Entraîner joueur.
- **Pré-conditions** : Néant.
- **Post-conditions** : La capacité est sélectionnée.
- **Cas général** : Le client choisit la capacité qu'il veut entraîner pour le joueur concerné.
- **Cas exceptionnels** : Néant.

2.1.3 Cas d'utilisation 3 : Gestion du domaine

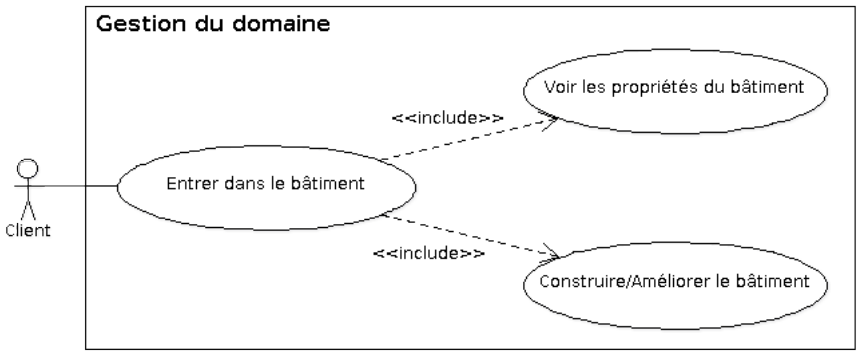

FIGURE 2.3: Cas d'utilisation 3 : Gestion du domaine

Une fois que l'utilisateur a rejoint sa partie ou en a créé une nouvelle, il peut gérer son domaine. Au commencement du jeu, ce domaine comprend un stade de petite taille, des infrastructures médiocres pour les entraînements et les soins à procurer aux joueurs, qui deviendront un beau centre d'entraînement et un hôpital à la pointe de la technologie, et également d'emplacements pour un fan shop et un centre de promotion (pour gagner des points d'action). Le client peut donc entamer des projets de construction et d'améliorer de bâtiments pour que ces derniers soient plus performants ou plus lucratifs.

Entrer dans le bâtiment

- **Relations avec d'autres cas d'utilisation** : Inclut Voir les propriétés du bâtiment et Construire/Améliorer le bâtiment.
- **Pré-conditions** : Néant.
- **Post-conditions** : Le client est dans le bâtiment.

- **Cas général** : Le client ïentredans le bâtiment et a la possibilité de lancer un projet de construction ou d'obtenir les informations concernant ce bâtiment.
- **Cas exceptionnels** : Néant.

Voir les propriétés du bâtiment

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Entrer dans le bâtiment.
- **Pré-conditions** : Néant.
- **Post-conditions** : Les informations du bâtiment s'affichent.
- **Cas général** : Le client souhaite avoir des renseignements sur le bâtiment dans lequel il est entré, s'affichent alors le niveau du bâtiment, le montant à payer et le nombre de points d'action nécessaires pour passer au niveau suivant et les caractéristiques particulières propres au bâtiment (comme par exemple le nombre de places s'il s'agit du stade).
- **Cas exceptionnels** : Néant.

Construire/Améliorer le bâtiment

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Entrer dans le bâtiment.
- **Pré-conditions** : Le client doit posséder suffisamment d'argent et de points d'action pour lancer le projet de construction ou d'amélioration. Il ne doit pas y avoir non plus un projet en cours sur ce bâtiment.
- **Post-conditions** : Le bâtiment est en projet pour une durée déterminée par son niveau. A la fin de ce laps de temps, le niveau du bâtiment sera incrémenté.
- **Cas général** : Le client souhaite améliorer le bâtiment dans lequel il est entré et est prêt à dépenser de l'argent et des points d'action pour ce faire.
- **Cas exceptionnels** : Néant.

2.1.4 Cas d'utilisation 4 : Enchères

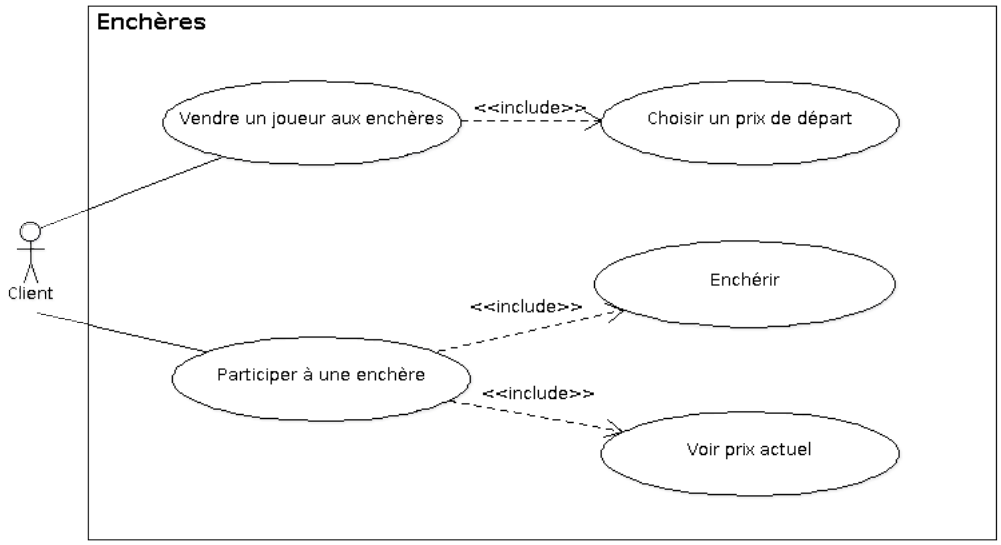

FIGURE 2.4: Cas d'utilisation 4 : Gestion de l'équipe

L'utilisateur n'étant pas obligé de se coltiner à vie son équipe de bras cassés, il peut acheter des joueurs mis en enchères par d'autres utilisateurs, ou même se débarasser des siens en les mettant en vente. Pour participer à une enchère, l'utilisateur doit faire une première offre, il est alors inscrit au tour suivant. A chaque enchère faite par un participant, le prix d'achat augmente de 10% du prix de départ. La vente se déroule par tour. A chacun de ces tours, les participants peuvent choisir d'enchérir ou non. S'ils enchérissent, ils sont inscrits pour le tour suivant, sinon, ils quittent la vente. Lorsqu'il n'y a qu'un seul participant qui enchérit au cours d'un tour, c'est lui qui remporte l'enchère. Si par contre aucun participant n'a enchéri avant la fin du tour actuel, c'est celui qui a enchéri en dernier au tour précédent qui remporte l'enchère. La transaction peut alors se faire.

Vendre un joueur aux enchères

- **Relations avec d'autres cas d'utilisation** : Inclut Choisir un prix de départ.
- **Pré-conditions** : Le joueur que le client souhaite vendre n'est pas bloqué par un entraînement ou un séjour à l'hôpital.
- **Post-conditions** : Le joueur mis en vente est bloqué pour toute la durée de l'enchère.
- **Cas général** : Le client souhaite vendre un joueur pour se débarasser de lui et gagner de l'argent.
- **Cas exceptionnels** :

- Si la vente du joueur réussit et que le client a désormais moins de 7 joueurs dans son équipe, il ne pourra plus faire de matches tant qu'il n'aura pas racheté des joueurs pour compléter son équipe.

Participer à une enchère

- **Relations avec d'autres cas d'utilisation** : Inclut Enchérir et Voir prix actuel.
- **Pré-conditions** : Néant.
- **Post-conditions** : Le client ne peut rien faire d'autre tant qu'il participe à l'enchère.
- **Cas général** : Le client est intéressé par le joueur et participe aux enchères. Il s'inscrit en faisant une première enchère.
- **Cas exceptionnels** :
  - Si le client participe à une enchère qu'il a lui-même créée, il est considéré comme un participant lambda et peut donc remporter l'enchère. S'il gagne, il se versera l'argent à lui-même, et achètera son propre joueur, ce qui est une perte de temps.

Choisir un prix de départ

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Vendre un joueur aux enchères.
- **Pré-conditions** : Néant.
- **Post-conditions** : Le prix de départ déterminera le prix réel de vente puisque chaque enchère fait augmenter le prix de 10% du prix de départ.
- **Cas général** : Le client choisit le prix de départ de l'enchère. Il peut ne pas respecter le montant estimé du joueur, mais c'est à ses risques et périls. Un prix de départ trop cher pourrait rebuter les acheteurs, et un prix de départ trop faible pourrait représenter une perte économique.
- **Cas exceptionnels** : Néant.

Enchérir

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Participer à une enchère.
- **Pré-conditions** : Le client n'a pas encore enchéri pendant le tour actuel.
- **Post-conditions** : Le client est inscrit pour le prochain tour de l'enchère.
- **Cas général** : Le client est toujours intéressé par le joueur et enchérit car il veut remporter l'enchère.
- **Cas exceptionnels** : Néant.

Voir prix actuel

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Participer à une enchère.
- **Pré-conditions** : Néant.
- **Post-conditions** : Le prix actuel de l'enchère est affiché.

- **Cas général** : Le client souhaite voir à combien s'élève le montant qu'il devrait débourser pour acheter le joueur s'il remportait l'enchère maintenant.
- **Cas exceptionnels** : Néant.

2.1.5 Cas d'utilisation 5 : Matches

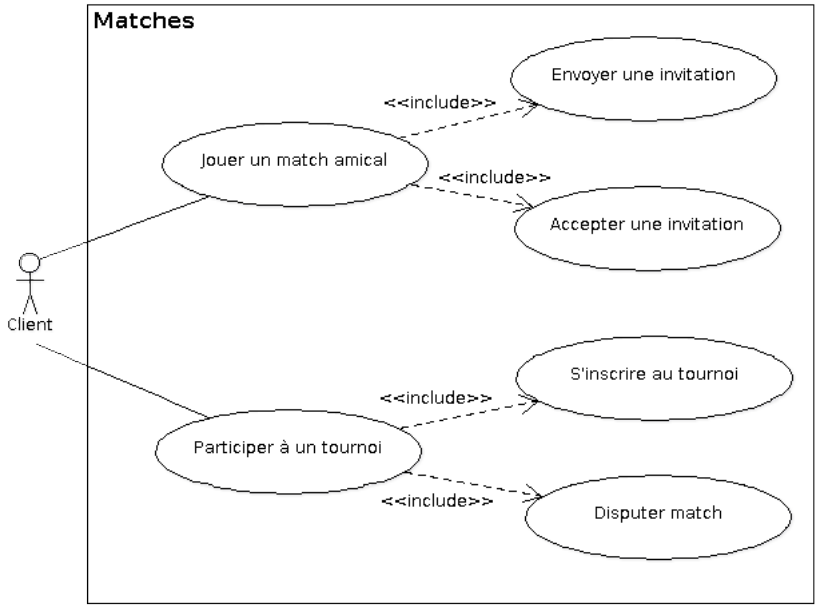

FIGURE 2.5: Cas d'utilisation 5 : Matches

L'utilisateur a la possibilité de mettre ses joueurs en action au cours de vrais matches de Quidditch. Pour cela, il peut soit s'inscrire à un tournoi, soit proposer un match amical à un autre utilisateur connecté. Dans le cadre d'un tournoi, les tours de matchs démarrent une fois que le bon nombre d'inscrits est atteint. L'utilisateur qui jouera le mieux, ou qui sera le plus chanceux, remportera le tournoi et les récompenses qui vont avec. Gagner un match amical offre également des avantages mais qui sont, bien entendu, moins conséquents.

Jouer un match amical

- **Relations avec d'autres cas d'utilisation** : Inclut Envoyer une invitation et Accepter une invitation.
- **Pré-conditions** : Néant.
- **Post-conditions** : Si le client remporte le match, ou qu'il y a match nul, il recevra une certaine somme d'argent en récompense.

- **Cas général** : Le client a envoyé une invitation qui a été acceptée par l'adversaire ou a accepté une invitation qui lui a été envoyée et va donc affronter un autre utilisateur. Le match commencera quand les deux utilisateurs auront désigné les joueurs et leur poste.
- **Cas exceptionnels** : Néant.

Participer à un tournoi

- **Relations avec d'autres cas d'utilisation** : Inclut S'inscrire au tournoi et Disputer match.
- **Pré-conditions** : Néant.
- **Post-conditions** : A chaque match gagné lors du tournoi, le client reçoit de l'argent en récompense.
- **Cas général** : Le client s'est inscrit à un tournoi (créé par l'administrateur). Lorsqu'il y a suffisament d'inscrits, le tournoi démarre tous les matchs du premier tour. Dès qu'un tour est terminé, le tour suivant commence (avec les joueurs ayant gagné leur match du tour précédent), jusqu'à ce qu'il n'y ait qu'un gagnant.
- **Cas exceptionnels** : Néant.

Envoyer une invitation

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Jouer un match amical.
- **Pré-conditions** : Le client possède suffisamment de points d'action pour envoyer une demande et il possède au moins 7 joueurs non bloqués dans son équipe.
- **Post-conditions** : Le client ne peut rien faire tant qu'il attend la réponse de l'autre utilisateur. Si l'adversaire accepte, les points d'action sont retirés et le match commence, sinon le client revient à sa session de jeu et aucun point d'action ne lui sera retiré.
- **Cas général** : Le client a accès à la liste des utilisateurs connectés et choisit celui contre qui il a envie de disputer un match amical.
- **Cas exceptionnels** : Néant.

Accepter une invitation

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Jouer un match amical.
- **Pré-conditions** : Le client a au moins 7 joueurs non bloqués dans son équipe.
- **Post-conditions** : Le match commence.
- **Cas général** : Le client a reçu une invitation d'un autre utilisateur connecté qui souhaite l'affronter au cours d'un match amical.
- **Cas exceptionnels** : Néant.

S'inscrire au tournoi

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Participer à un tournoi.
- **Pré-conditions** : Néant.
- **Post-conditions** : Le client est inscrit à ce tournoi et ne pourra donc pas s'inscrire à un autre tournoi.
- **Cas général** : Le client a accès à la liste des tournois pour lesquels il manque encore des participants et a la possibilité de s'inscrire à l'un deux.
- **Cas exceptionnels** : Néant.

Disputer match

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Participer à un tournoi.
- **Pré-conditions** : Le client a au moins 7 joueurs non bloqués dans son équipe.
- **Post-conditions** : Si le client remporte le match, il est qualifié pour le tour d'après et gagne de l'argent.
- **Cas général** : Le jour de l'affrontement dans le cadre du tournoi est arrivé et le client va jouer son match contre son adversaire.
- **Cas exceptionnels** :
  - Le match était le dernier du tour du tournoi. Le tour suivant commence alors.
  - Si le résultat est un match nul, le gagnant est alors celui qui a proposé le match nul.

2.1.6 Cas d'utilisation 6 : Tour d'un match

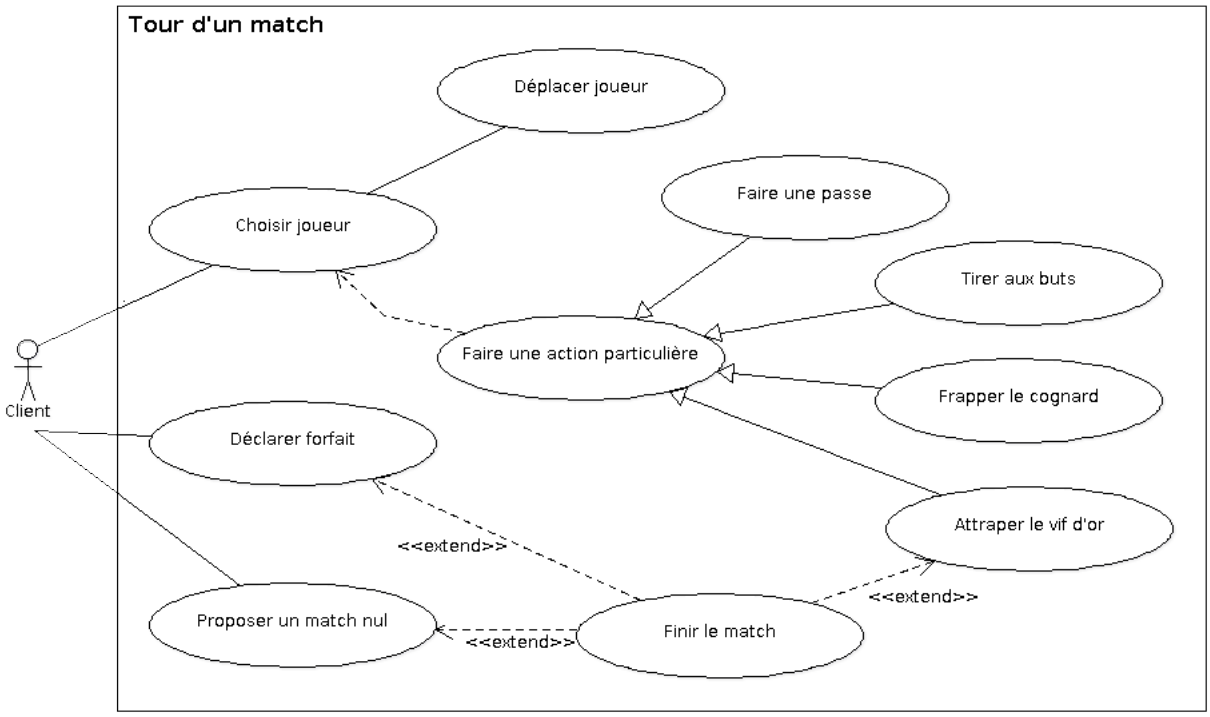

FIGURE 2.6: Cas d'utilisation 6 : Tour d'un match

Les matches se jouent au tour par tour. A chaque tour, les clients qui s'affrontent sont invités à indiquer les actions (déplacement ou action particulière) à faire pour chacun de leur joueur.

Choisir joueur

- **Relations avec d'autres cas d'utilisation** : Précède Déplacer joueur et Faire une action particulière (ce use case dépendant de celui-ci : en effet, les actions possibles changent en fonction du joueur choisi).
- **Pré-conditions** : Le client n'a pas déjà choisi quoi faire pour ce joueur.
- **Post-conditions** : Le client ne pourra plus choisir quoi faire pour ce joueur.
- **Cas général** : Le client va, pour chaque tour, indiquer à chacun de ses joueurs ce qu'il doit faire ; se déplacer ou effectuer une action particulière dépendant du joueur (faire une passe, etc.). Le joueur peut également rester sur place et ne rien faire (c'est en quelque sorte un déplacement de longueur 0).
- **Cas exceptionnels** : Néant.

Déplacer joueur

- **Relations avec d'autres cas d'utilisation** : Lié à Choisir joueur
- **Pré-conditions** : Néant.
- **Post-conditions** : Le joueur sélectionné a été déplacé ou est resté sur place.
- **Cas général** : Le client sélectionne un joueur et choisit où celui-ci doit se déplacer sur le terrain ou le laisse sur place. Le client est informé de l'ensemble des destinations possibles pour le joueur sélectionné.
- **Cas exceptionnels** :
  - Le client ne choisi aucune direction pour le joueur sélectionné. Le joueur reste donc sur place.

Faire une action particulière

- **Relations avec d'autres cas d'utilisation** : Dépend de Choisir joueur. Généralise Faire une passe, Tirer aux buts, Frapper le cognard et Attraper le vif d'or.
- **Pré-conditions** : Néant.
- **Post-conditions** : Le joueur a une action particulière à accomplir, ses chances de réussir dépendent de ses capacités.
- **Cas général** : Le client choisit le joueur et l'action que celui-ci doit réaliser. Cette action dépend du poste du joueur.
- **Cas exceptionnels** : Néant.

Faire une passe

- **Relations avec d'autres cas d'utilisation** : Spécialise Faire une action particulière.
- **Pré-conditions** : Le joueur qui va faire la passe doit tenir le souafle et doit être un poursuiveurs ou un gardiens .
- **Post-conditions** : Que la passe soit réussie ou ratée, le joueur ne possèdera plus le souafle.
- **Cas général** : Le client choisit l'endroit de destination de la passe. Les endroits possibles dépendent des compétences du joueur qui a le souafle.
- **Cas exceptionnels** :
  - La balle entre en collision avec une autre balle ou un joueur. Son déplacement s'arrête.

Tirer aux buts

- **Relations avec d'autres cas d'utilisation** : Spécialise Faire une action particulière.
- **Pré-conditions** : Le poursuiveur qui veut tirer doit tenir le souafle et se trouver dans l'alignement d'un des buts.
- **Post-conditions** : Qu'il y ait but ou non, le joueur ne tiendra plus le souafle.

- **Cas général** : Le client essaie de marquer un but en indiquant au poursuiveur détenant le souafle vers quel but il doit tirer. Si le tir est réussi, l'équipe qui vient de marquer gagne 10 points.
- **Cas exceptionnels** :
  - La balle entre en collision avec une autre balle ou un joueur. Son déplacement s'arrête.

Frapper le cognard

- **Relations avec d'autres cas d'utilisation** : Spécialise Faire une action particulière
- **Pré-conditions** : Le joueur qui va essayer de frapper un cognard doit être un batteur et doit se situer près du cognard.
- **Post-conditions** : Le cognard est projeté dans la direction voulue.
- **Cas général** : Le code client indique dans quelle direction le batteur doit frapper le cognard.
- **Cas exceptionnels** :
  - Le batteur rate son coup. Rien ne se passe.

Attraper le vif d'or

- **Relations avec d'autres cas d'utilisation** : Spécialise Faire une action particulière et est étendu par Finir le match
- **Pré-conditions** : Un des attrapeurs a repéré le vif d'Or.
- **Post-conditions** : Si l'attrapeur réussit son coup, le match est terminé. S'il échoue, le match continue.
- **Cas général** : Un des attrapeurs est arrivé à proximité du vif d'or. Il essaiera alors de l'attraper (la réussite de l'action dépend des qualités de l'attrapeur et de sa distance par rapport au vif d'or). S'il réussit à l'attraper, son équipe gagne 150 points et le match est terminé.
- **Cas exceptionnels** :
  - L'attrapeur échoue. Le vif d'or reste libre.

Déclarer forfait

- **Relations avec d'autres cas d'utilisation** : Est étendu par Finir le match.
- **Pré-conditions** : L'action se déroule au cours d'un match.
- **Post-conditions** : Le match est terminé. Le client qui n'a pas déclaré forfait a gagné.
- **Cas général** : Un des client souhaite mettre fin au match en déclarant forfait (équivaut à une défaite 150-0, peu importe le score lorsqu'il déclare forfait).
- **Cas exceptionnels** :
  - Si un client se déconnecte au cours d'un match, on considérera qu'il a déclaré forfait.

Proposer un match nul

- **Relations avec d'autres cas d'utilisation** : Est étendu par Finir le match.
- **Pré-conditions** : L'action se déroule au cours d'un match.
- **Post-conditions** : Si l'adversaire accepte la proposition, le match sera terminé, sinon le match continue.
- **Cas général** : Un des client souhaite mettre fin au match en proposant un match nul à l'adversaire.
- **Cas exceptionnels** :
  - Il s'agit d'un match de tournoi. Celui ayant demandé la fin du match est considéré comme perdant.

Finir le match

- **Relations avec d'autres cas d'utilisation** : Etend Attraper le vif d'or, Déclarer forfait et proposer un match nul.
- **Pré-conditions** : Néant.
- **Post-conditions** : Il y a un club gagnant et un club perdant ou deux ex-aequos.
- **Cas général** : Le match va se terminer, le score est figé et les résultats vont déterminer ce que va gagner ou perdre (argent, fans) chaque client.
- **Cas exceptionnels** :
  - Un des clubs a déclaré forfait. L'autre club est gagnant.

2.1.7 Cas d'utilisation 7 : Points d'action

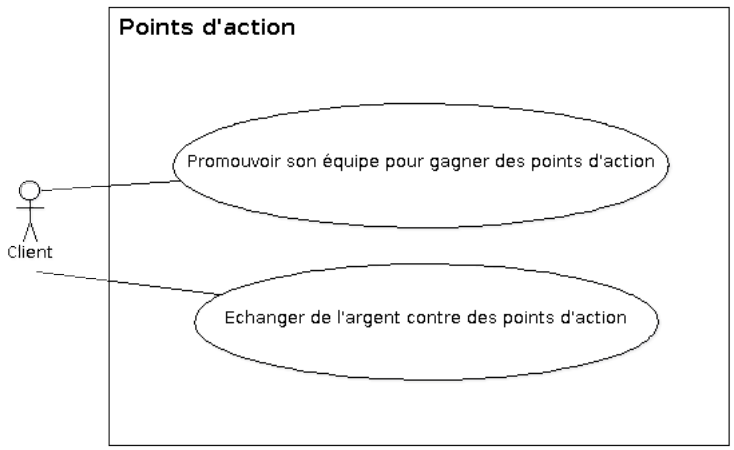

FIGURE 2.7: Cas d'utilisation 7 : Points d'action

L'utilisateur possède un certain nombre de points d'action qui limite ce qu'il peut faire. En effet, chaque action de gestion, qu'elle ait un coût financier ou non, ne peut

être exécutée que si le client possède suffisamment de points d'action. Puisque ces points sont vite dépensés, il faut bien que l'utilisateur puisse en récupérer.

Promouvoir son équipe pour gagner des points d'action

- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Néant.
- **Post-conditions** : Le client ne peut rien faire d'autre tant qu'il n'aura pas quitté sa campagne de promotion. Une fois celle-ci terminée (choix du client), il gagnera des points d'action.
- **Cas général** : Le client veut récupérer gratuitement des points d'action et va donc promouvoir son équipe. Il s'agit d'une attente active : au plus longtemps il attend, au plus de points d'action il pourra récupérer. Ce nombre dépend également du niveau du centre de promotion.
- **Cas exceptionnels** :
  - Le client se déconnecte en pleine campagne. Il n'aura gagné aucun point d'action.

Echanger de l'argent contre des points d'action

- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Le client doit posséder suffisamment d'argent.
- **Post-conditions** : De l'argent est retiré au client et des points d'action lui sont redistribués.
- **Cas général** : Le client souhaite gagner immédiatement des points d'action et est prêt à mettre la main au portefeuille pour ce faire. Le taux de change or-point d'action est préfixé et est le même pour tous les utilisateurs.
- **Cas exceptionnels** : Néant.

2.1.8 Cas d'utilisation 8 : Session de l'administrateur

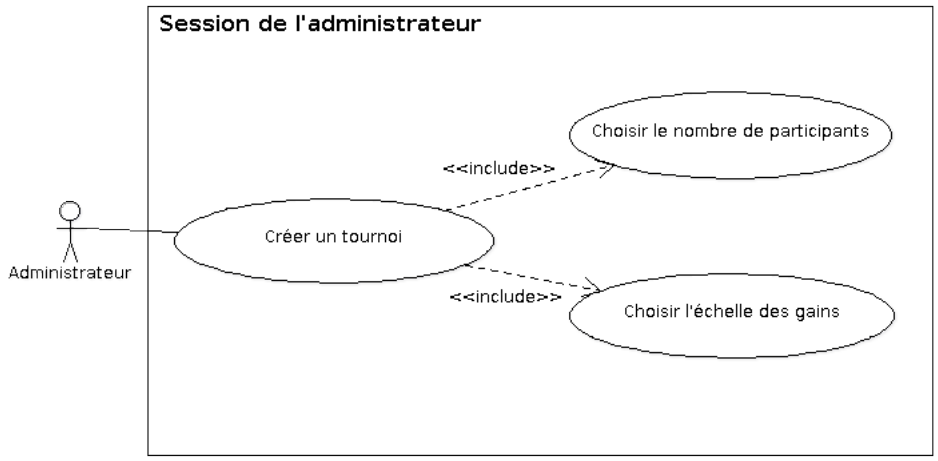

FIGURE 2.8: Cas d'utilisation 8 : Session de l'administrateur

L'administrateur est un utilisateur particulier qui n'a pas de session de jeu comme les autres mais qui peut créer des tournois.

Créer un tournoi

- **Relations avec d'autres cas d'utilisation** : Inclut Choisir le nombre de participants et Choisir l'échelle des gains.
- **Pré-conditions** : Néant.
- **Post-conditions** : Le tournoi est créé.
- **Cas général** : L'administrateur décide de créer un tournoi pour que les utilisateurs puissent y participer.
- **Cas exceptionnels** : Néant.

Choisir le nombre de participants

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Créer un tournoi.
- **Pré-conditions** : Néant.
- **Post-conditions** : Néant.
- **Cas général** : L'administrateur choisit combien de participants, et donc combien de tours, comprendra le tournoi. Il ne peut bien entendu par choisir n'importe quel nombre.
- **Cas exceptionnels** :
  - L'administrateur entre un nombre qui n'est pas une puissance de deux. La création est annulée.

Choisir l'échelle des gains

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Créer un tournoi.
- **Pré-conditions** : Néant.
- **Post-conditions** : Néant.
- **Cas général** : L'administrateur choisit l'échelle des gains ; c'est-à-dire un montant qui représente la somme gagnée pour le premier tour du tournoi. La croissance de ce montant est fixée et les montants pour les tours suivants sont donc facilement calculés.
- **Cas exceptionnels** : Néant.

2.2 Exigences non fonctionnelles

Pour le confort de l'utilisateur, il faudra une interface graphique très intuitive, les menus étant attachés à des bâtiments, leur sélection par la souris ouvrira ces menus. De même, la gestion d'un match devra être aisée, malgré le nombre d'actions à choisir sur un tour. Le plaisir de l'utilisateur apporté par le jeu dépendra du rythme de la succession de ces tours.

2.3 Exigences de domaine

L'expérience immersive de l'utilisateur doit être connectée à l'univers fixé, à savoir la saga d'Harry Potter. Cela signifie, d'une part, qu'il ne faut pas contrevenir aux droits d'auteur portant sur les oeuvres qui décrivent cet univers, et d'autre part, qu'il faut y ressembler suffisament pour que l'utilisateur, familier de ce même univers, y retrouve les éléments le composant.

3 Besoins du système

3.1 Exigences fonctionnelles

3.1.1 Cas d'utilisation 1 : Quidditch Manager 2014

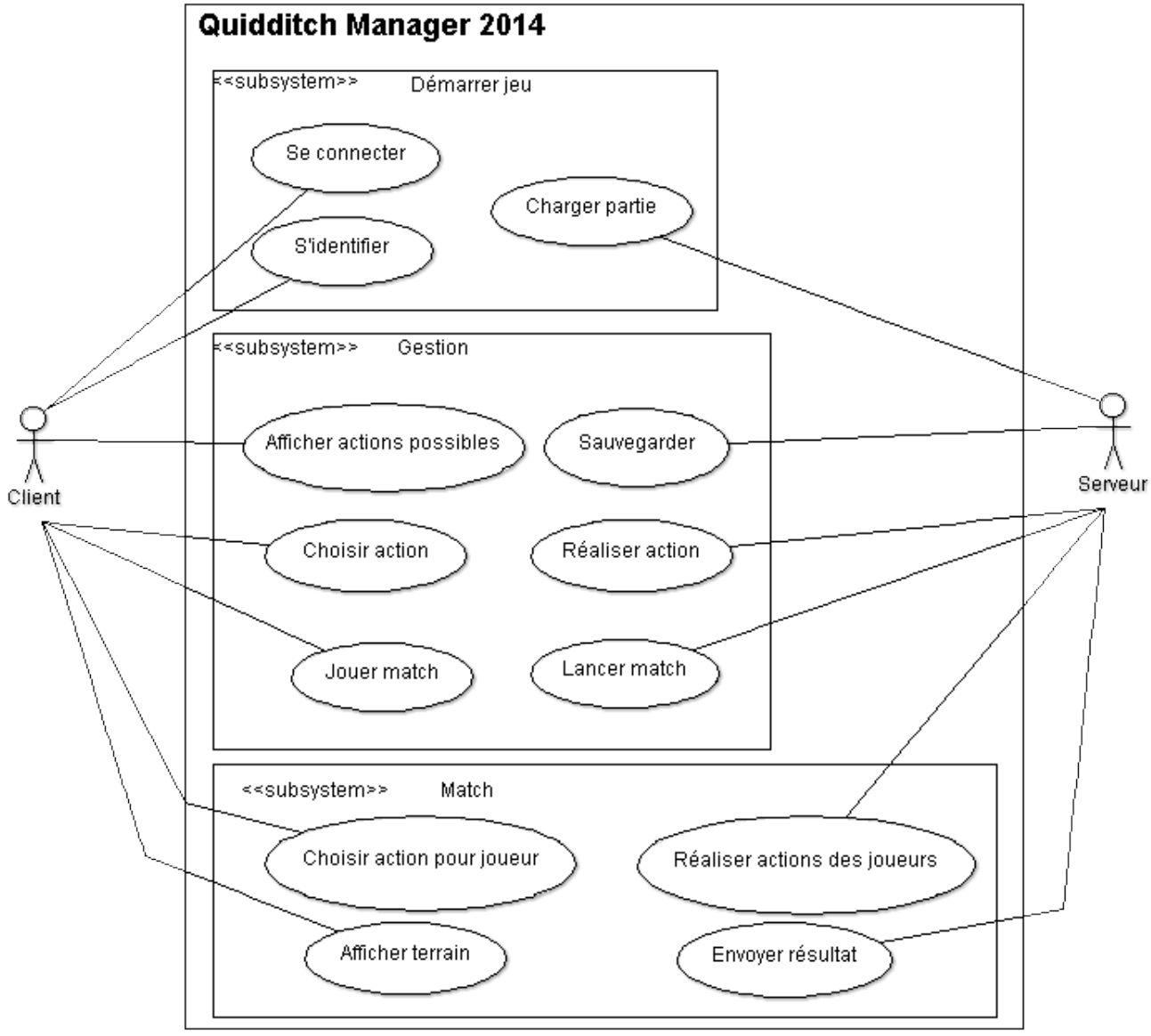

FIGURE 3.1: Cas d'utilisation 1 : Quidditch Manager 2014

Ce diagramme représente le fonctionnement de base du jeu avec la répartition des "tâches" au niveau serveur et client.

Se connecter

- **Acteur** : Client.
- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Le serveur doit être connecté.
- **Post-conditions** : Client et serveur sont reliés par le réseau.
- **Cas général** : Le client se connecte au serveur du jeu pour pouvoir jouer.
- **Cas exceptionnels** :
  - La connexion au serveur échoue. Le programme le signale à l'utilisateur. Ceci termine ce use case.

S'identifier

- **Acteur** : Client.
- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Le client est connecté au serveur.
- **Post-conditions** : Le client a accès à sa partie.
- **Cas général** : Le client crée une nouvelle partie ou en charge une existante pour s'identifier auprès du serveur. Si le client charge une partie existante, le serveur va récupérer le fichier de sauvegarde approprié, sinon, il va en créer un nouveau.
- **Cas exceptionnels** :
  - Le client entre un identifiant ou un mot de passe déjà utilisé. Le programme le signale à l'utilisateur et lui redemande de s'identifier.

Charger partie

- **Acteur** : Serveur.
- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : La connexion et l'authentification d'un client se sont déroulés sans encombre.
- **Post-conditions** : La partie est chargée.
- **Cas général** : Le serveur va charger les informations de la partie à laquelle va accéder le client.
- **Cas exceptionnels** :
  - Le client ne possède pas encore de partie. Le serveur lui en crée une.

Sauvegarder

- **Acteur** : Serveur.
- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Des informations relatives à la partie d'un client ont été modifiées et nécessitent d'être sauvées.

- **Post-conditions** : Les changements sont sauvegardés dans le fichier de sauvegarde dédié au client.
- **Cas général** : Le serveur sauvegarde les informations de la partie. L'état actuel de la partie du client est sauvegardée dans un fichier de sauvegarde.
- **Cas exceptionnels** : Néant.

Afficher actions possibles

- **Acteur** : Client.
- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Le client est connecté au serveur.
- **Post-conditions** : Néant.
- **Cas général** : Le client affiche tout ce qu'il est possible de faire d'un point de vue gestion.
- **Cas exceptionnels** : Néant.

Choisir action

- **Acteur** : Client.
- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Le client est connecté au serveur.
- **Post-conditions** : La requête a été envoyée au serveur.
- **Cas général** : Le client a choisi une action de gestion à effectuer (amélioration bâtiment, entraînement, publicité, achat, vente, gestion d'équipe). Il doit maintenant attendre que le serveur exécute cette action.
- **Cas exceptionnels** :
  - L'envoi échoue. Le programme le signale à l'utilisateur.

Réaliser action

- **Acteur** : Serveur.
- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Client et serveur sont connectés.
- **Post-conditions** : La requête a été traitée par le serveur.
- **Cas général** : Le client a choisi de faire une action de gestion (amélioration bâtiment, entraînement, publicité, achat, vente, gestion d'équipe). Le serveur va se charger de faire les modifications liées à cette action.
- **Cas exceptionnels** :
  - La réception échoue. Le programme le signale à l'utilisateur.

Jouer match

- **Acteur** : Client.
- **Relations avec d'autres cas d'utilisation** : Néant.

- **Pré-conditions** : La date et le moment de la journée correspondent à un match prévu dans le championnat.
- **Post-conditions** : Ce match est retiré du calendrier des matches.
- **Cas général** : On va quitter la partie gestion pour lancer le match. Avant de pouvoir commencer le match, le client doit choisir la liste des joueurs qui seront sur le terrain.
- **Cas exceptionnels** : Néant.

Lancer match

- **Acteur** : Serveur.
- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Les deux clients devant s'affronter sont connectés.
- **Post-conditions** : Néant.
- **Cas général** : Le serveur va maintenant gérer le match et l'aspect gestion ne sera plus accessible tant que le match ne sera pas terminé.
- **Cas exceptionnels** : Néant

Choisir action pour joueur

- **Acteur** : Client.
- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Le joueur sélectionné est encore libre de faire une action.
- **Post-conditions** : L'action choisie est réalisée par ce joueur lors du tour.
- **Cas général** : Le client choisit l'action que devra réaliser le joueur lors du tour. Cela peut-être un déplacement et/ou une action particulière comme passer le souaffle ou tirer aux buts. Le client doit faire cela pour chacun de ses joueurs ou indiquer au serveur qu'il a fini de choisir ses actions (s'il souhaite par exemple que des joueurs ne fassent rien).
- **Cas exceptionnels** :
  - Le client ne donne aucune instruction au joueur. Le joueur reste sur place.

Réaliser action des joueurs

- **Acteur** : Serveur.
- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Cette action se déroule dans le cadre d'un match.
- **Post-conditions** : La situation sur le terrain a évolué.
- **Cas général** : A chaque tour, le serveur attend de recevoir les actions que les clients ont choisies pour chacun de leurs joueurs. Une fois l'ensemble de ces actions reçues, le serveur les exécute puis envoie la nouvelle situation aux clients. Les clients affichent alors le terrain et redemandent les actions à exécuter aux utilisateurs.
- **Cas exceptionnels** :
  - Si un des clients ne s'est pas connecté pour le match, le serveur n'attendra pas d'actions de la part de ce client. Ses joueurs suivront une stratégie automatique

minimaliste prise en charge par le serveur.

Afficher terrain

- **Acteur** : Client.
- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Le client est connecté au serveur et un match a lieu.
- **Post-conditions** : Néant.
- **Cas général** : Le client reçoit les informations relatives au terrain de la part du serveur et l'affiche pour permettre au client de facilement visualiser la situation actuelle du jeu.
- **Cas exceptionnels** : Néant.

Envoyer résultat

- **Acteur** : Serveur.
- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Le match est terminé.
- **Post-conditions** : Le client dispose des informations à afficher après un match.
- **Cas général** : Le match est terminé. Le serveur va calculer combien le Manager a gagné (en fonction des bâtiments fanshop (match à domicile) ou buvette (match en extérieur)). En fonction de la situation de fin de match, victoire ou défaite, le serveur va également déterminer combien d'argent et de popularité le Manager a gagné ou perdu. Toutes ses informations sont envoyées au client.
- **Cas exceptionnels** : Néant.

3.2 Exigences non fonctionnelles

3.2.1 Le réseau

Au niveau du réseau, le problème de la synchronisation n'est pas préoccupant, car le client ne sera qu'une interface d'affichage et d'interaction. Pour ce qui est de la quantité d'informations échangées, elle ne devrait pas être très importante pour autant, les éléments à afficher et leur diversité étant relativement réduits.

Il faudra veiller à ce que l'interaction avec l'utilisateur soit transparente au niveau du serveur, à travers une classe de message générique, que le programme client instanciera de la même manière, qu'il tourne à travers une interface graphique ou à travers le terminal.

3.2.2 Performance

Pour ce qui est de la performance, la localisation de tous les calculs sur la machine serveur implique une centralisation de la puissance de calcul. D'un autre côté, le client ne nécessitera que très peu de puissance de son côté pour pouvoir jouer.

3.2.3 Sécurité

La sécurité d'une partie sera assurée par l'identification : seul un utilisateur identifié peut charger une partie, et la sienne uniquement.

3.2.4 Environnement d'exécution

L'environement d'exécution étant fixé (les machines des salles informatiques du NO), le code se limitera à l'utilisation des librairies C++ standards. Le système d'exploitation visé est GNU/Linux.

3.2.5 Choix d'une librairie graphique

Qt est un framework d'interface graphique orienté objet et développé en C++ par la PME norvégienne Trolltech. La première version gratuite fut mise à disposition du public en mai 1995[1].
Il offre des composants d'interface graphique (widgets), d'accès aux données, de communication réseau, de gestion des threads.
Il permet la portabilité des applications qui n'utilisent que ses composants par simple recompilation du code source. Les environnements supportés sont les Unix (dont Linux) qui utilisent le système graphique X Window System, Windows et Mac OS X, mais aussi Android, iOS et QNX (BlackBerry).
Il peut s'intégrer dans des développements réalisés dans la plupart des langages de programmation modernes, d'Ada à Python.
Ses librairies sont accompagnées de plusieurs outils de développement, parmi lesquels :

- Un IDE ;
- Un concepteur d'interface graphique ;
- Des plugins pour Eclipse et Visual Studio.

Qt est notamment connu pour être la bibliothèque sur laquelle repose l'environnement graphique KDE, l'un des environnements de bureau les plus utilisés dans le monde Linux ; des logiciels commerciaux comme Adobe Photoshop Album, Autodesk Maya et Mathematica, et des logiciels gratuits omniprésents comme Skype, VLC media player et VirtualBox, l'utilisent également.
La popularité de Qt (des milliers de clients et des dizaines de milliers de programmeurs) est un point important dans le choix de la bibliothèque graphique. Elle permet en effet d'assurer :

- Sa pérennité et son adaptation aux évolutions technologiques ;
- Sa fiabilité et la qualité de son support ;
- Une communauté d'utilisateurs actifs sur de nombreux forums d'entraide ;
- Une documentation abondante et de qualité.

3.3 Design et fonctionnement du système

3.3.1 Le jeu sur terminal

Nous allons ici décrire comment se déroule une partie sur terminal. Nous pouvons tout d'abord distinguer 2 parties différentes, la gestion et le match. **Gestion** : La partie gestion du jeu permet donc au Manager, le client, de gérer une équipe ainsi qu'un domaine. Ce domaine se compose de plusieurs bâtiments, ayant chacun leur particularité et avantage. Le Manager peut donc se balader au travers des différents menus qui lui permettent d'accéder aux informations de ses joueurs (leurs capacités, etc) et de ses bâtiments et d'effectuer des actions de gestion telles qu'entraîner ou soigner un joueur, participer à une enchère ou améliorer un bâtiment.
Match : A tout moment, s'il possède au moins 7 joueurs disponibles, le Manager peut envoyer une demande de match amical à un autre client connecté. Si la réponse de ce dernier est positive, le match débutera. Ce match de Quidditch se déroule au tour par tour. Chacun de leur côté, les adversaires choisissent les mouvements qu'ils veulent que leurs joueurs fassent, sans savoir ce qu'a préparé l'autre manager. Et ainsi de suite jusqu'à une situation de fin de partie.

3.3.2 Implémentation

Dans cette partie sera expliquée l'implémentation générale du jeu ; de quelles façons les fonctionnalités ont été mises en place, les choix faits, etc. Des diagrammes de classes viendront détailler ces explications.

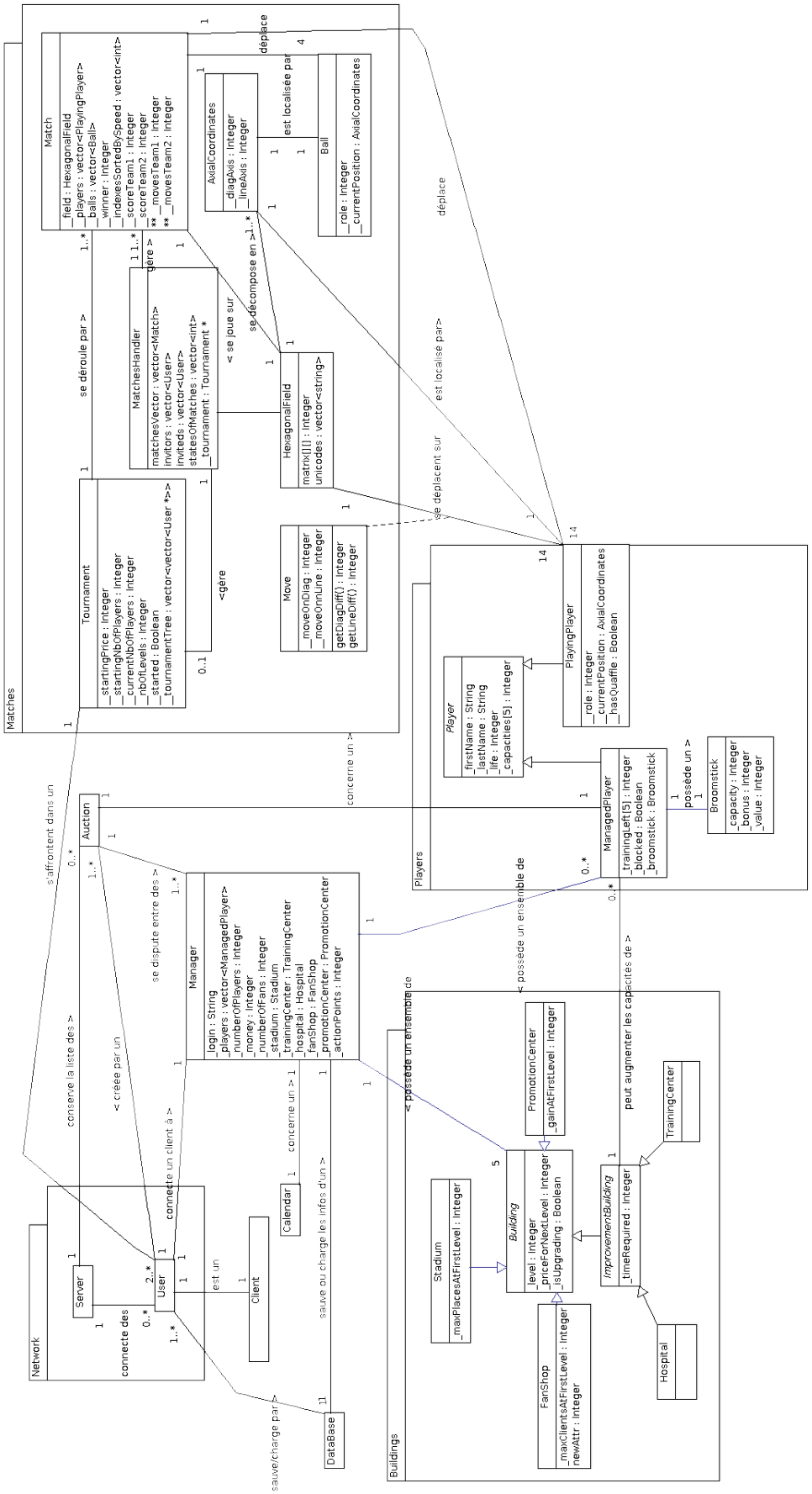

FIGURE 3.2: Diagramme de classes global

Gestion générale

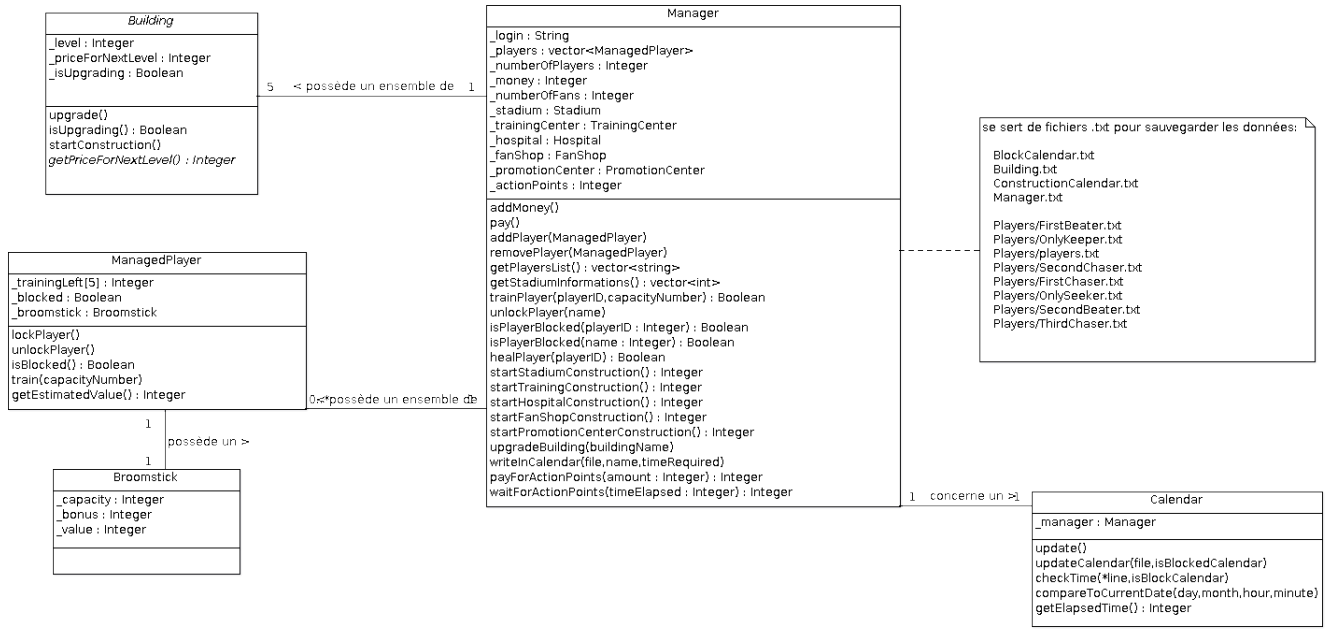

FIGURE 3.3: Diagramme des classes impliquées dans la gestion générale

Un manager est le ¨personnage virtuelq̈u'incarne un client. Il est le propriétaire d'un domaine et d'une équipe de quidditch. Au départ du jeu, il possède quelques bâtiments sur son domaine, et sept joueurs de qualité moyenne, si pas médiocre. Pour améliorer ses bâtiments, ou les capacités de ses joueurs, le manager doit dépenser des points d'action, de l'argent, et attendre. L'attente en temps réel est gérée par la classe Calendrier.

Bâtiments

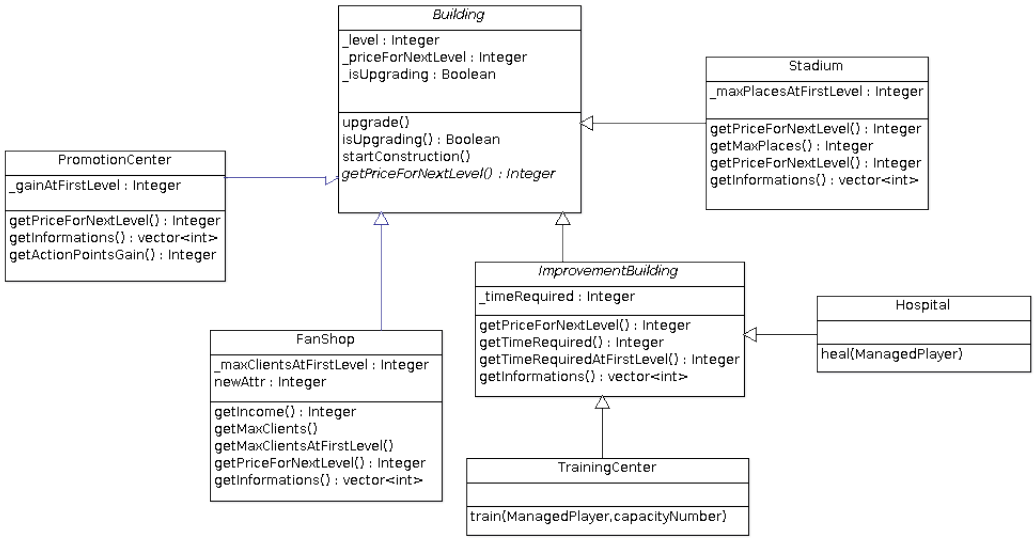

FIGURE 3.4: Diagramme des classes pour les bâtiments

Le domaine du manager contient 5 emplacements pour chacun des bâtiments qu'il peut posséder. Au début du jeu, 4 des 5 bâtiments sont déjà au niveau, seul le fan shop n'est pas encore construit.

Les 5 bâtiments sont :

- Le stade : Il s'agit du lieu dans lequel se joueront les matches. La capacité en spectateurs ainsi que le nombre de fans du manager déterminent l'argent gagné lorsqu'un match a lieu dans le stade. La capacité augmente avec les niveaux.
- Le centre d'entraînement : C'est ici que viennent s'entraîner les joueurs pour améliorer leurs capacités. La durée d'une session d'entraînement, et de son blocage, diminue quand le niveau du bâtiment augmente.
- L'hôpital : C'est ici que le manager envoie ses joueurs blessés (suite à un match) pour qu'ils soient soignés. La durée des soins, et du blocage, diminue quand le niveau du bâtiment augmente.
- Le fan shop : Il s'agit d'un magasin qui vend des produits aux couleurs de l'équipe du manager. A chaque match à domicile, un certain nombre de fans pourra se rendre dans le magasin et y faire des achats. La recette dépend donc de ce nombre, qui est déterminé par le niveau du bâtiment.
- Le centre de promotion : C'est ici que le manager peut venir récupérer des points d'action gratuitement. Au lieu de payer, il attend un certain temps pour recevoir des points actions. Le niveau du bâtiment détermine combien de points d'action il gagne en attendant x minutes (ce nombre de minutes étant lui-même défini dans le

code, il peut facilement être changé).

Joueurs

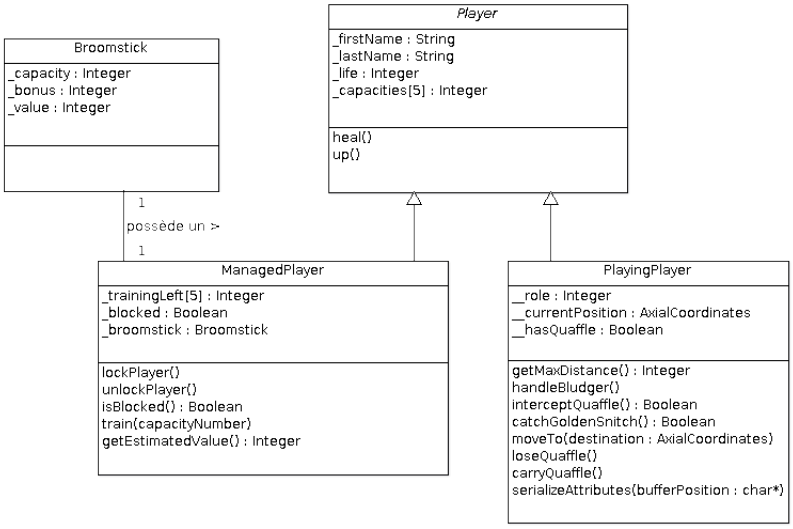

FIGURE 3.5: Diagramme des classes pour les joueurs

Du point de vue de l'implémentation, les joueurs sont différenciés s'ils sont en phase de gestion ou en train de jouer un match, d'où les deux classes ManagedPlayer, pour la gestion, et PlayingPlayer, pour les matches. La seule différence entre les deux est que le bonus lié au balai (actuellement toujours à 0) est absorbé dans les capacités du joueur telles qu'elles sont vues pendant un match.

Un joueur se distingue des autres par son nom, choisi au hasard à la création du joueur. Les différents postes que peuvent occuper les joueurs, à savoir attrapeur, gardien, poursuiveur et batteur, ne sont pas prédéterminés. Le manager peut choisir à quel poste faire jouer ses joueurs avant chaque match. Néanmoins, dans un souci d'équité, lors de la création de l'équipe de base (nouvelle partie), les joueurs par défaut attribués au client présentent des capacités axées sur un poste particulier.

Les capacités, ou caractéristiques des joueurs, sont les suivantes :
- La vitesse, qui détermine la longueur maximale des déplacements lors d'un match
- La force, qui détermine l'aptitude des batteurs à frapper un cognard
- La précision, qui intervient dans les passes, les tirs, etc.
- Les réflexes, pour attraper le vif d'or, intercepter le souafle, etc.
- La résistance, qui détermine la "vie" du joueur, ou combien de coups de cognard il peut recevoir avant d'être incapable de poursuivre le match.

Les joueurs possèdent également un balai (c'est en effet plus pratique pour voler) qui, s'il est de qualité, booste un de ces 5 attributs (fonctionnalité non implémentée mais envisagée).

Pour pouvoir améliorer les capacités de ses joueurs, un manager peut les envoyer s'entraîner. Pour augmenter une capacité et passer au niveau suivant, le joueur devra réaliser un certain nombre de sessions d'entraînement, ce nombre étant déterminé par le niveau actuel de la capacité concernée.

Enfin, lorsque la vie du joueur n'est pas optimale, le manager peut l'envoyer à l'hôpital se faire soigner.

Sauvegarde et chargement

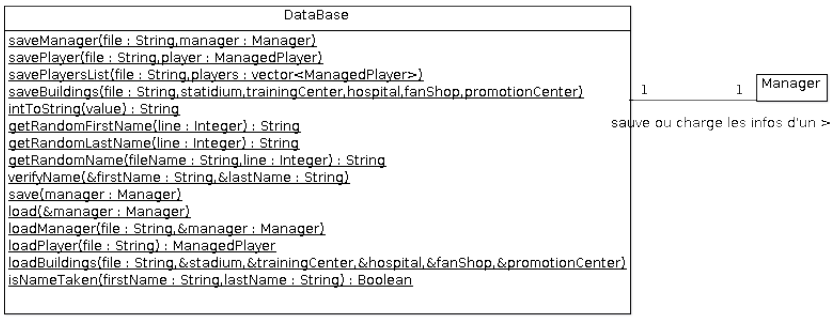

FIGURE 3.6: Diagramme de classes pour les sauvegardes/chargements

La structure des fichiers de sauvegarde se présente comme suit :

A chaque client est associé un dossier de sauvegarde portant le nom de son compte; son login. Dans ce dossier se trouvent des fichiers .txt : un se nommant selon son login (donc un format login.txt), buildings.txt, blockCalendar.txt et constructionCalendar.txt

Le fichier login.txt contient les informations relatives aux attributs de l'objet Manager, c'est-à-dire le nombre de joueurs possédés, l'argent, le nombre de fans et le nombre de points d'action que le client a. Le format du fichier est donc :

- Nombre de joueurs
- Argent

- Nombre de fans
- Nombre de points d'action

Le fichier buildings.txt contient les informations relatives à chaque bâtiment du client ; le stade, le centre d'entraînement, l'hôpital, le fan shop et le centre de promotion. Il y a 4 informations pour chaque bâtiment : son niveau actuel, le prix pour la construction (passage du niveau 0 au niveau 1), la valeur au niveau 1 de l'attribut particulier à chaque bâtiment et un indicateur de projet (0 si le bâtiment est libre, 1 si on a lancé une construction ou une amélioration et qu'elle n'est pas encore terminée). Le format du fichier est donc :

- Niveau du stade
- Prix de construction du stade
- Nombre de places dans le stade au niveau 1
- Indicateur de projet
- Niveau du centre d'entraînement
- Prix de construction du centre d'entraînement
- Durée du blocage en tours au niveau 1
- Indicateur de projet
- Niveau de l'hôpital
- Prix de construction de l'hôpital
- Durée du blocage en tours au niveau 1
- Indicateur de projet
- Niveau du fan shop
- Prix de construction du fan shop
- Nombre de clients max dans le fan shop au niveau 1
- Indicateur de projet
- Niveau du centre de promotion
- Prix de construction du centre de promotion
- Nombre de points d'action gagnés par tranche de temps au niveau 1
- Indicateur de projet

Le fichier blockCalendar.txt contient les informations concernant les blocages des joueurs quand ceux-ci sont à l'hôpital ou à l'entraînement. Chaque ligne représente un blocage et a donc le nom du joueur concerné, la date de début du blocage ainsi que la durée en minutes de ce blocage. Le format d'une ligne du fichier est donc :

- Prénom Nom#Date de début (au format jour :mois :heure :minute)#Durée du blocage#

Le fichier constructionCalendar.txt contient les informations concernant les projets de construction ou d'amélioration des bâtiments. Chaque ligne correspond à un projet et a donc le nom du bâtiment concerné, la date de début de construction ou d'amélioration et la durée en minutes du projet. Le format d'une ligne du fichier est donc :

- Nom du bâtiment#Date de début (au format jour :mois :heure :minute)#Durée du projet#

En plus de ces 4 fichiers, le dossier de sauvegarde du client contient un sous-dossier nommé Players. Ce sous-dossier contient un fichier players.txt qui est une liste de tous les autres fichiers contenus dans ce sous-dossier. Ces autres fichiers « PrénomNom.txt » correspondent chacun à un joueur de l'équipe du client et possèdent toutes les informations relatives à ce joueur. Le fichier players.txt permet donc de savoir quels fichiers il faut lire quand on va charger une partie.

Un fichier de joueur, PrénomNom.txt contient donc les informations du joueur, c'est-à-dire ses capacités, l'avancement de ses entraînements, etc. Le format est le suivant :

- Prénom
- Nom
- Vitesse
- Force
- Précision
- Réflexe
- Résistance
- Vie
- Nombre de sessions d'entraînement à faire avant d'améliorer la vitesse
- Nombre de sessions d'entraînement à faire avant d'améliorer la force
- Nombre de sessions d'entraînement à faire avant d'améliorer la précision
- Nombre de sessions d'entraînement à faire avant d'améliorer le réflexe
- Nombre de sessions d'entraînement à faire avant d'améliorer la résistance
- 1 si le joueur est bloqué ou 0 sinon
- Indice de la capacité pour laquelle le balai offre un bonus
- La valeur du bonus offert par le balai

Enchères

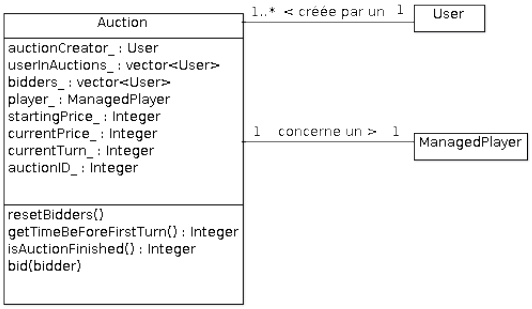

FIGURE 3.7: Diagramme de classes pour les enchères

Les enchères se déroulent au tour par tour. A chaque tour, les managers désireux de poursuivre la vente doivent enchérir. S'ils ne le font pas dans le temps imparti, ou qu'ils souhaitent quitter l'enchère, ils devront attendre la fin du tour avant d'avoir à nouveau accès aux menus de jeu. Le dernier tour est atteint quand il n'y a plus qu'un seul manager qui enchérit, il gagne alors la vente. Pour gérer le temps imparti par tour, la solution adoptée est l'utilisation de threads. Le thread principal lance à chaque début de tour un thread sur une méthode qui enregistre le choix du manager (enchérir ou non). Après avoir lancé ce thread secondaire, le thread principal s'endort pour une durée déterminée. A son réveil, il invite cordialement l'autre thread à s'arrêter. Le tour est fini et les managers ne peuvent plus enchérir. On répète donc cela jusqu'à ce qu'il n'y ait plus qu'un seul manager à avoir enchéri.

Matches

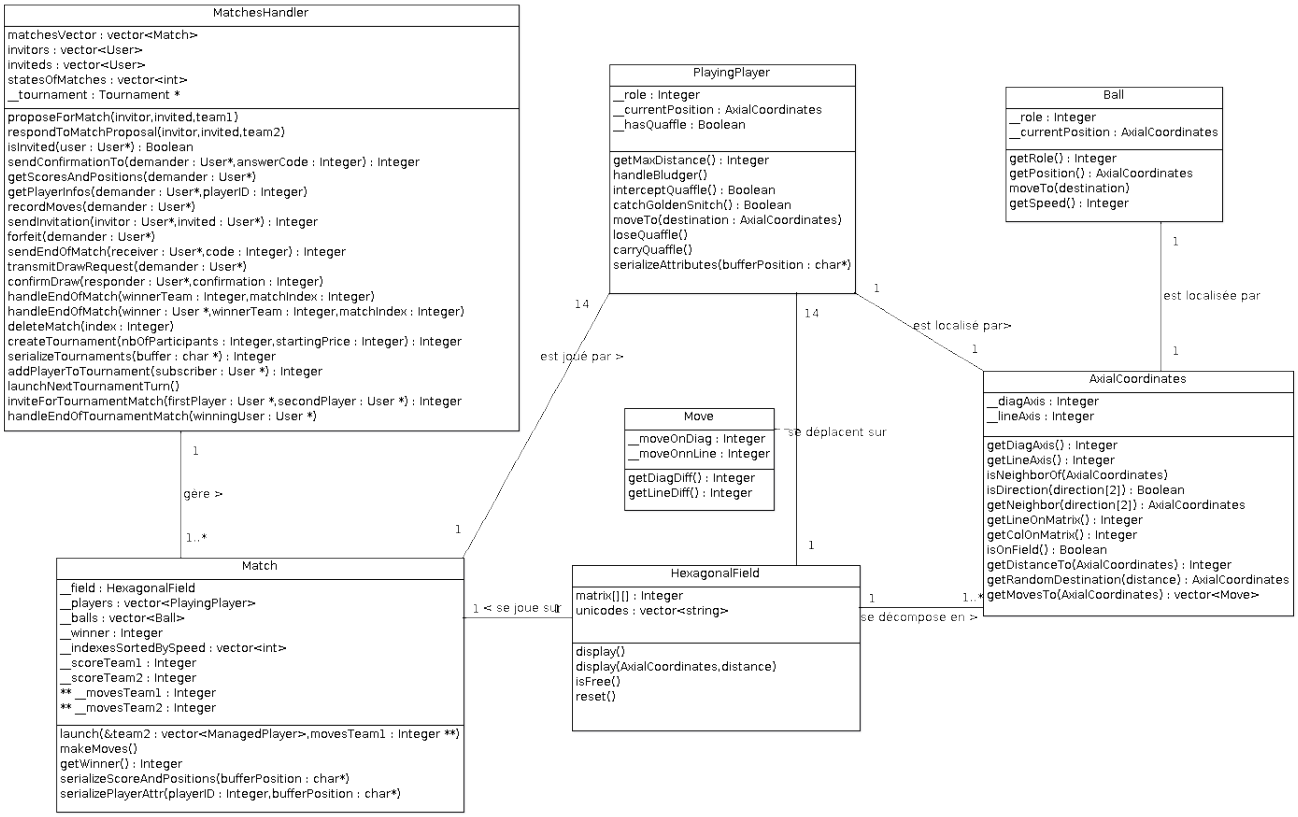

FIGURE 3.8: Diagramme de classes pour les matches

Le match relie deux clients différents dans un affrontement sur un terrain où se déplacent à la fois des joueurs et des balles, les joueurs pouvant faire se déplacer les balles. Le terrain est représenté par un hexagone allongé et respecte le format ̈case hexagonale; c'est-à-dire qu'il y a 6 cases de destination adjacentes à une case donnée (si elle n'est pas sur les bords). Un client peut participer à deux ̈typesde match, soit un match amical, en invitant un autre joueur ou en étant invité, soit un match de tournoi, qui offre des récompenses plus importantes. Au début de chaque match, le client choisit quels joueurs seront sur le terrain, et à quel poste ils joueront. Le déroulement des matches consiste en du tour par tour.

Tournois

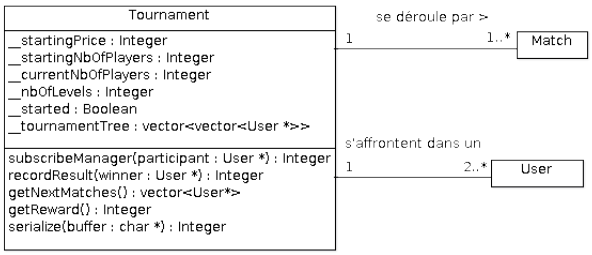

FIGURE 3.9: Diagramme de classes pour les tournois

A compléter.

Réseau

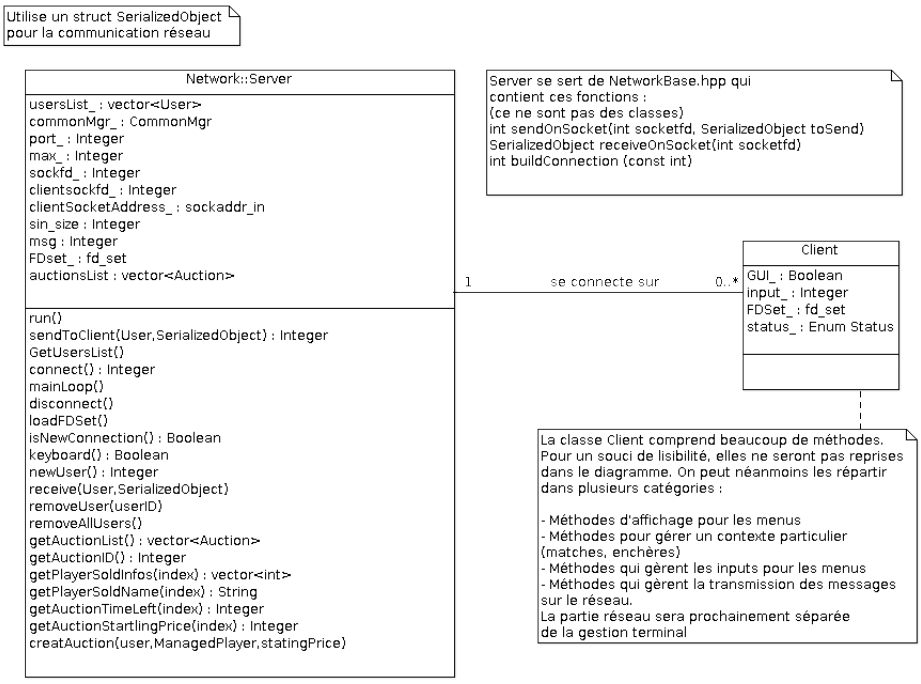

FIGURE 3.10: Diagramme de classes pour le serveur

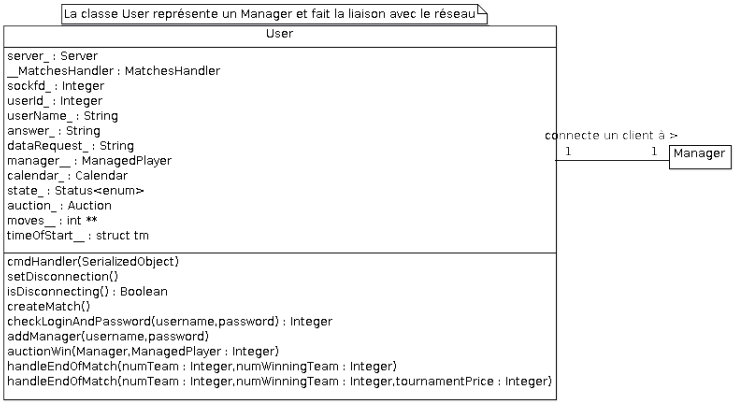

FIGURE 3.11: Diagramme de classes pour les utilisateurs sur le réseau

La relation avec le client est gérée par une classe principale, qui va assurer la communication à travers une classe de messages (un *struct* puisque la base réseau consiste en des appels systèmes C). C'est cette classe principale qui va gérer les alternances entre le côté gestion et le côté match. Le match nécessite une connexion entre deux clients, elle est donc très différente de la gestion à ce niveau-là aussi.

Interface graphique

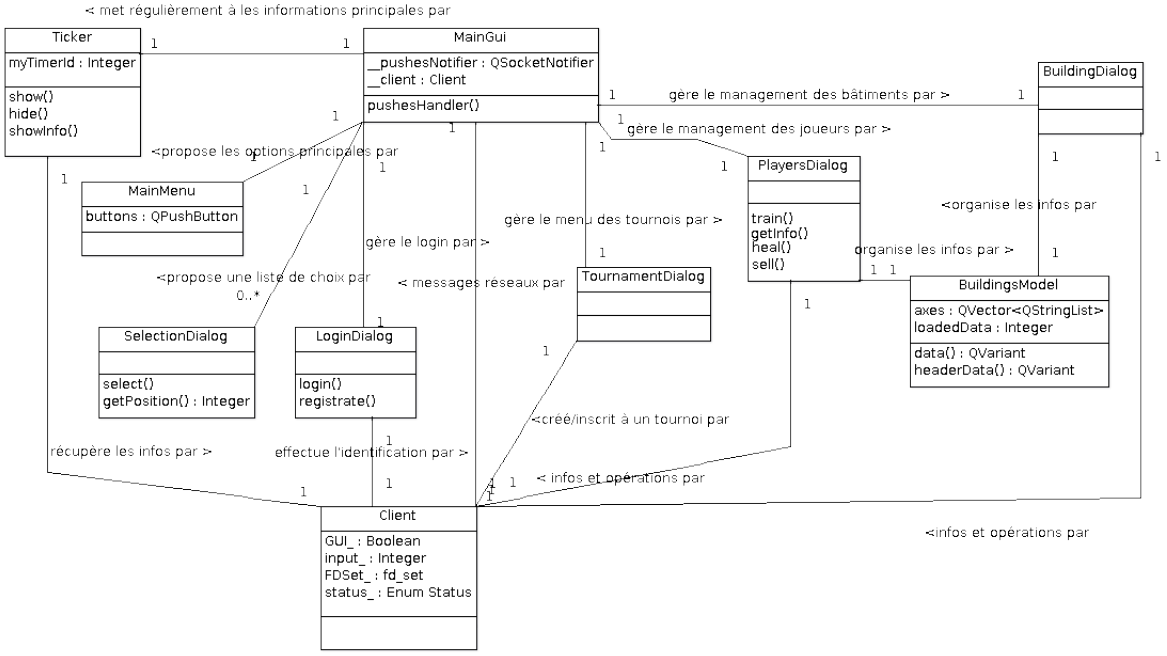

FIGURE 3.12: Diagramme des classes impliquées dans la GUI

Une fenêtre principale fait appel à différentes sous-fenêtres pour des interactions particulières. Si une fenêtre a besoin de communiquer avec le serveur, elle le fera à travers un pointeur de la classe Client, la fenêtre principale instanciant une classe client à sa construction.

Pour une description plus élaborée de la partie graphique, le lecteur se reportera utilement au § 3.4.

3.3.3 Le réseau

L'interface réseau doit permettre une communication bidirectionnelle :
- des clients chargés de communiquer chacun avec un User : affichage d'informations et enregistrement de commandes ;
- un serveur chargé de partager de l'information entre les clients en la centralisant et en la conservant sur disque.

La logique applicative peut être répartie entre ces deux entités, comme elle peut être concentrée sur le serveur (le client se comporte alors comme un browser sans scripts) ou sur le client (le serveur peut se limiter aux fonctions de gestionnaire de fichiers ou de bases de données et de données temporaires en mémoire centralisée).
Entre les deux, l'interface réseau, comme tout middleware, ne doit pas connaître les formats internes des messages échangés entre clients et serveur. Le respect de ce principe doit permettre d'assurer à la fois l'indépendance entre le fonctionnel et le technique, et une séparation du code réalisé pour couvrir ces deux aspects du projet.
En général, le modèle client-serveur [2] est serveur-centrique :

- les clients ne communiquent qu'avec le serveur, pas entre eux ; le partage d'informations entre les clients se fait donc via le serveur, selon un modèle de tableau d'affichage (tableau noir), partagé par les/certains clients ;
- la communication et les dialogues qui s'en suivent se font à l'initiative des clients.

Ce deuxième principe ne permet cependant pas de répondre à tous les besoins exprimés pour ce projet : les clients doivent pouvoir être invités dans des dialogues ; ils doivent donc être en mesure, lorsqu'ils ne sont pas occupés par une activité, de recevoir des messages non sollicités ; nous avons donc dû revoir notre design initial en conséquence, tel qu'il apparaîtra ci-après.

Echange de messages serveur-client

Les messages échangés entre la partie serveur et la partie client du projet sont transmis selon un principe de sérialisation et désérialisation qui permet de faire transiter sur le réseau des messages totalement différents qu'on identifie par un en-tête. Cet en-tête permet au code qui reçoit ce message de savoir comment le traiter.

On constate donc la présence de deux structures de séquencement fort similaires, en forme de rateau :

- un input ;
- le choix d'une séquence d'instruction (une méthode) en fonction de cet input ;
- un output.

Algorithm 1 Traitement séquentiel partagé par le serveur et le client

Require: socket(s) ouvert(s)
  écoute sur les sockets ouverts et sur le flux d'entrée
  if input sur flux d'entrée **then**
    traitement : a l'initiative
  else {message sur le socket}
    traitement : sollicité
  end if

Synopsis du serveur

- Au démarrage, le serveur crée une instance de la classe *MatchesHandler*, destinée à gérer l'interconnexion de clients au travers du serveur dans le cas des matchs ;
- Le serveur se met en attente (fonction select()) d'une activité sur son clavier (pour commander l'arrêt du serveur par exemple ou obtenir la liste des utilisateurs connectés...), sur son socket destiné à recevoir les nouvelles connexions et sur les sockets des clients déjà connectés (voir diagrammes) ;
- Lors d'une nouvelle connexion, le serveur crée une instance de la classe User ; il y a donc un objet User par client ;
- Tout nouveau message reçu par le serveur est confié à la méthode *commandHandler* de l'objet de type User, qui assure le dispatching vers le module fonctionnel concerné, en se basant et mettant à jour la variable d'état state ;
- Cela assure un contrôle global des activités du client (et de l'arrêt de celles-ci). Remarquons que cela pourrait permettre d'entreprendre plusieurs activités en parallèle.

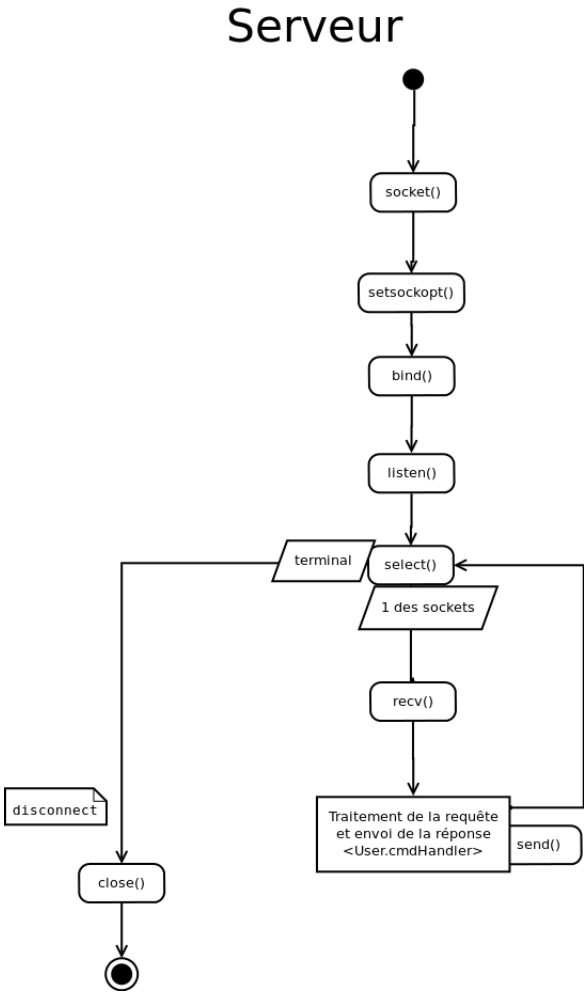

FIGURE 3.13: Diagramme d'activité : cycle de vie du serveur

Aucune des méthodes du serveur n'étant bloquante pendant plus d'une fraction de seconde (peu d'I/O) et compte tenu des délais très courts pour la mise en œuvre, nous avons opté à ce stade pour une solution mono-thread ; travailler avec un seul thread supprime la nécessité de devoir gérer l'accès concurrent aux données (en mémoire ou sur disque) et la communication entre threads (arrêt du serveur par exemple) : une solution multi-threads rendrait la gestion de ressources communes par les fonctions applicatives plus complexe (sérialisation de sections critiques), tout en étant beaucoup plus gourmande en ressources CPU et présentant des risques de deadlocks (tests plus délicats à réaliser !). Pour la gestion des time-out (dans le cas d'une enchère uniquement jusqu'à présent), cela est actuellement géré au niveau client par un thread de cadencement (qui ne tourne que le temps d'un tour), mais cela pourra évoluer vers une utilisation du paramètre temporel de l'appel système *select*.

Si les messages sont de longueur fixe prévisible, un buffer de taille fixe peut être utilisé

de part et d'autre. Sinon (par exemple pour une liste), deux solutions sont possibles :

- Envoyer une série de messages de taille fixe ;
- Utiliser de part et d'autre des zones de mémoire allouées dynamiquement et envoyer tout en un seul message, après un premier message indiquant au client la taille du buffer.

Jusqu'à présent nous allouons un buffer de taille fixe.

Synopsis du client

Comme le serveur, le client utilise la fonction select() afin de pouvoir capter indifféremment des messages d'origine différente, en l'occurrence en provenance du clavier ou du socket de communication, ceci pour pouvoir recevoir des messages non sollicités dans une solution mono-thread.

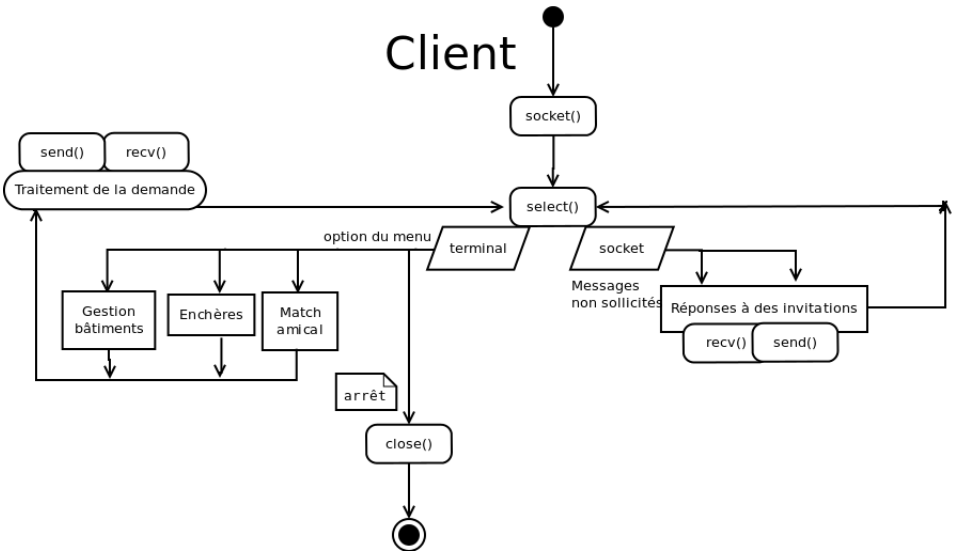

FIGURE 3.14: Diagramme d'activité : cycle de vie du client

La disponibilité aux messages non sollicités ne peut exister que lorsque le client n'est pas dans un contexte verrouillé, c'est-à-dire dans les menus principaux ou au début du tour d'un match.

Lorsque le manager choisit une activité, le client entre dans un dialogue avec à la fois le serveur et le manager, sans qu'il puisse être interrompu par un message non sollicité ; d'ailleurs, le serveur ne va pas lui en envoyer, puisqu'il connaît l'état dans lequel se trouve le client.

Le serveur travaillant en mode répétitif (pas de parallélisme), la variable d'état suffit donc à contrôler l'accès aux sections critiques que sont les différentes activités.

De même, lorsque le client reçoit un message non sollicité, il entre dans un dialogue avec à la fois le serveur et le manager, sans qu'il puisse être interrompu par un autre message non sollicité ou par une autre demande du manager au clavier.

Exemples

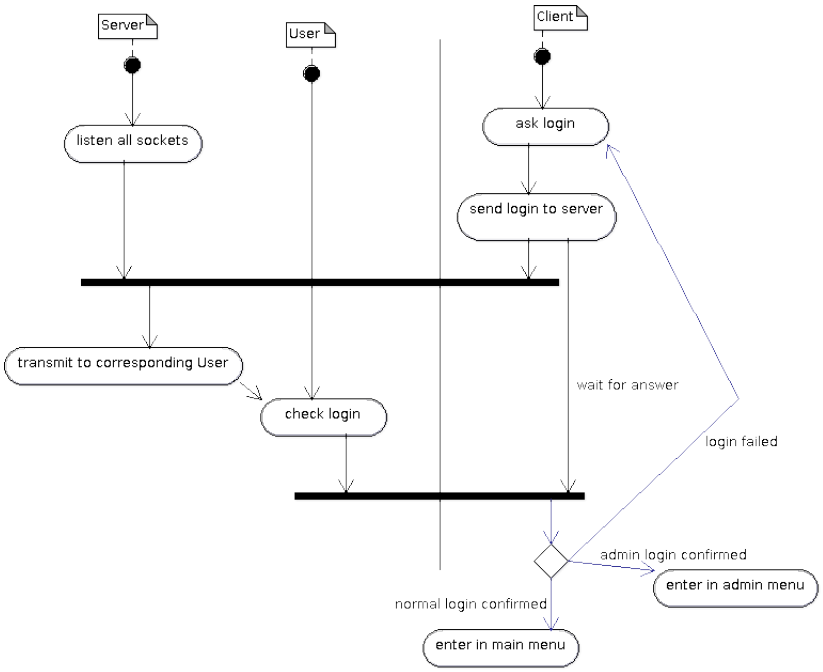

FIGURE 3.15: Exemple d'interaction par diagramme d'activité : identification

Cas où le client prend l'initiative de la communication : le code client récupère les identifiants et les envoie au serveur. Celui-ci transmet la requête à l'instance User correspondante qui va traiter la demande et communiquer la confirmation au client. En fonction de cette confirmation, le code client fera évoluer son contexte et affichera le menu adapté.

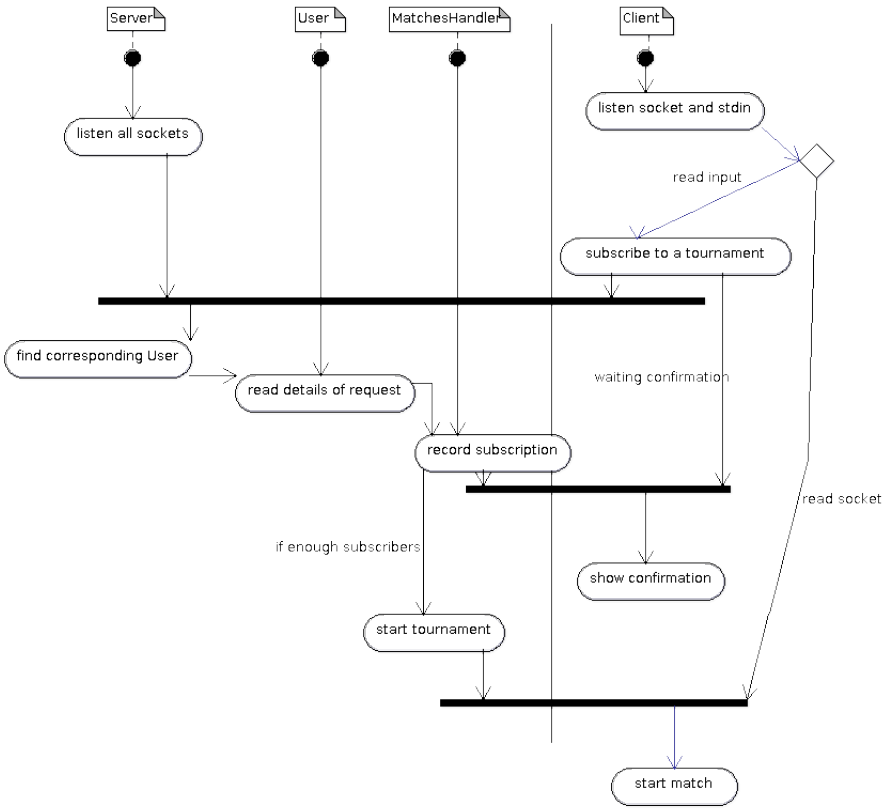

FIGURE 3.16: Exemple d'interaction par diagramme d'activité : démarrage d'un tournoi

L'inscription et le démarrage d'un tournoi impliquent les deux modes du client : l'initiative et la sollicitation. L'inscription se fait par l'input reçu sur le flux d'entrée, par lequel l'utilisateur donne ses instructions. L'enregistrement de cette inscription est similaire au processus d'identification, à la différence qu'une instance commune à tous les *Users* se charge du traitement et de la réponse.

Quand le nombre d'inscrits requis est atteint, le serveur va inviter les clients à jouer leur match. Dans ce cas, le serveur prend l'initiative d'une communication vers le client et c'est le client qui va devoir confirmer en envoyer l'équipe de joueurs choisie.

Diagramme de séquence

Voici un diagramme de séquence qui illustre les relations entre différents objets du code serveur et du code client, dans le cas d'un match amical.

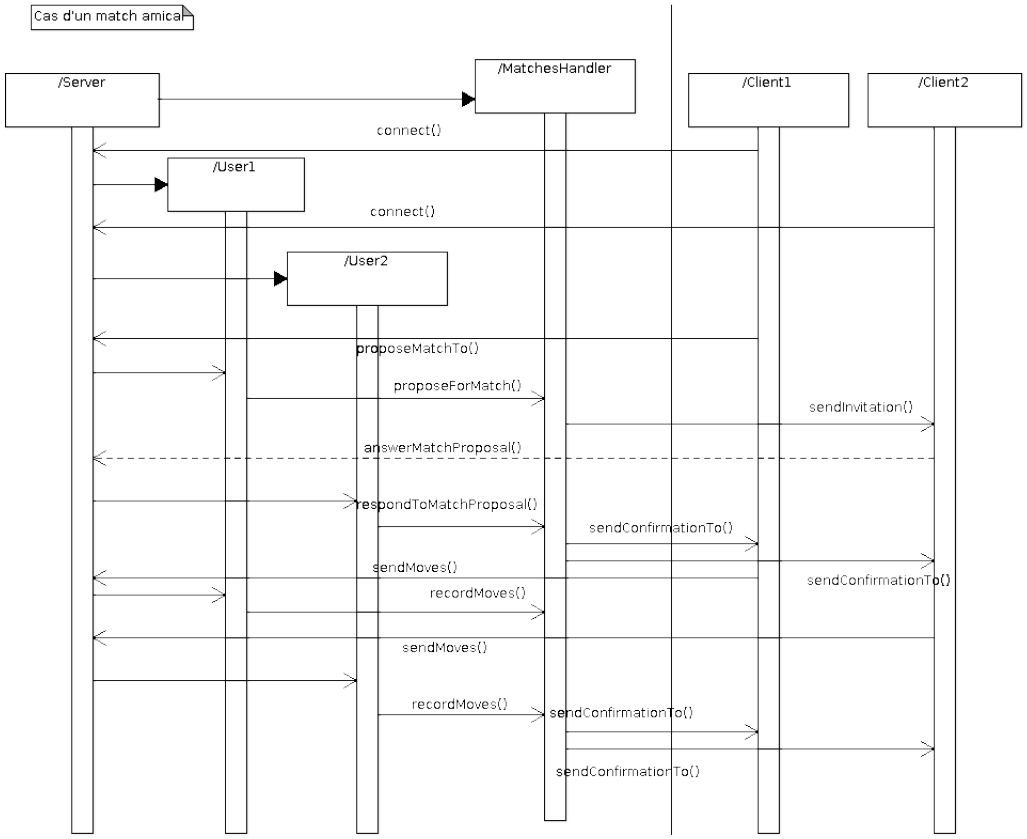

FIGURE 3.17: Diagramme de séquence : démarrage d'un match amical

À la connexion d'un client, le serveur instancie un User, classe chargé d'interpréter les messages reçus sur le socket de communication correspondant. Si cette requête concerne la gestion d'un match, alors le User transmet l'information au matchesHandler, qui a lui-même été instancié par le serveur à son initialisation et dont chaque User reçoit un pointeur. L'invitation d'un autre utilisateur est bloquante, le serveur ne répondant que lorsque l'utilisateur invité aura lui-même répondu. La réception d'une invitation du côté client implique une écoute du socket, qui se fait en même temps que l'écoute du flux d'entrée grâce à l'appel système *select* . Une gestion du contexte permet de savoir comment réagir lors d'un input sur le flux d'entrée, tandis que l'en-tête des messages reçus du serveur permet de savoir comment réagir lors d'un push (message non attendu) du serveur vers le client.

Une fois le match démarré, le code client se charge de remplir une matrice d'entiers caractérisant une liste de déplacements ou d'actions, qu'il envoie au serveur. Le premier client à répondre est bloqué le temps que le deuxième client ne réponde. Une gestion de time-out pourra être intégrée plus tard.

3.3.4 Diagramme de composants

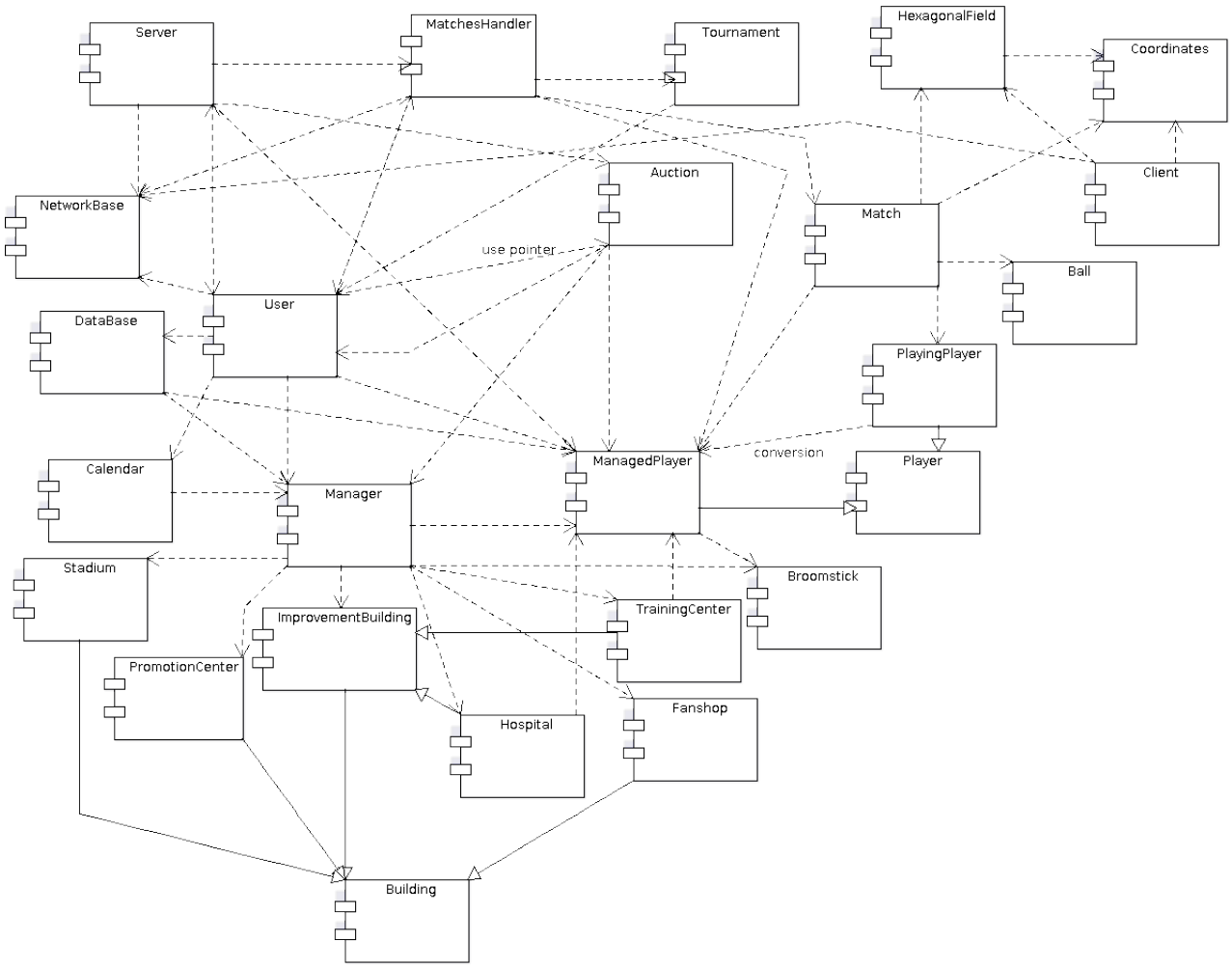

FIGURE 3.18: Diagramme de composants

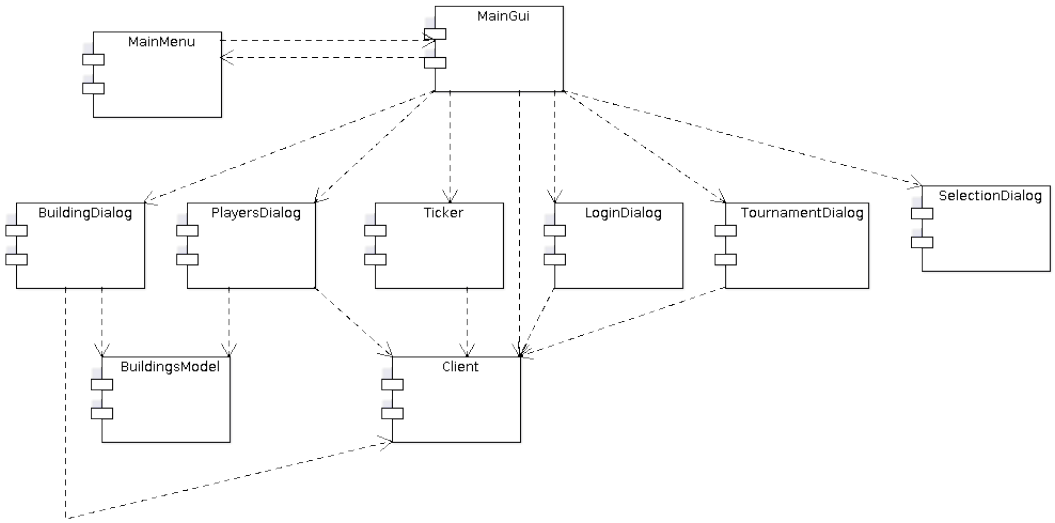

FIGURE 3.19: Diagramme simplifié de composants pour la partie graphique uniquement

3.4 Le client graphique

3.4.1 Le login

Le client graphique s'ouvre sur un dialogue de login permettant à l'utilisateur de se connecter au serveur, soit avec un nom d'utilisateur existant, soit en se créant une nouvelle identité *(checkbox "New User")*[1]. A ce moment, la fenêtre du jeu n'est pas encore visible.
Si l'utilisateur ne souhaite pas se connecter ("Quit"), il se voit proposer une fenêtre avec des menus réduits : "File" lui permettant de tenter à nouveau de se connecter ou, au contraire, d'arrêter le programme, ainsi qu'un menu "Help".

3.4.2 Les menus et les boutons

En cas de réussite du login ou de l'enregistrement, le serveur fournit au client le rôle de l'utilisateur : manager ou administrateur ; ce dernier est réservé à l'utilisateur "admin" (mode de passe "admin") prédéfini dans le fichier central d'identification.
En fonction de ce rôle, la fenêtre centrale de gestion *(classe "MainGui" héritant de QMainWindow : figure 3.20)*, présente une barre de menus différente.
Le manager dispose également de boutons *(classe "MainMenu")* activant les fonctions associées aux points principaux de la barre de menus. Ce rôle déterminera également de façon dynamique l'affichage de certains boutons dans le dialogue de gestion des tournois.

1. Les détails techniques de mise en œuvre sont mis entre parenthèses. Un mode d'emploi s'obtiendra en supprimant ces parties.

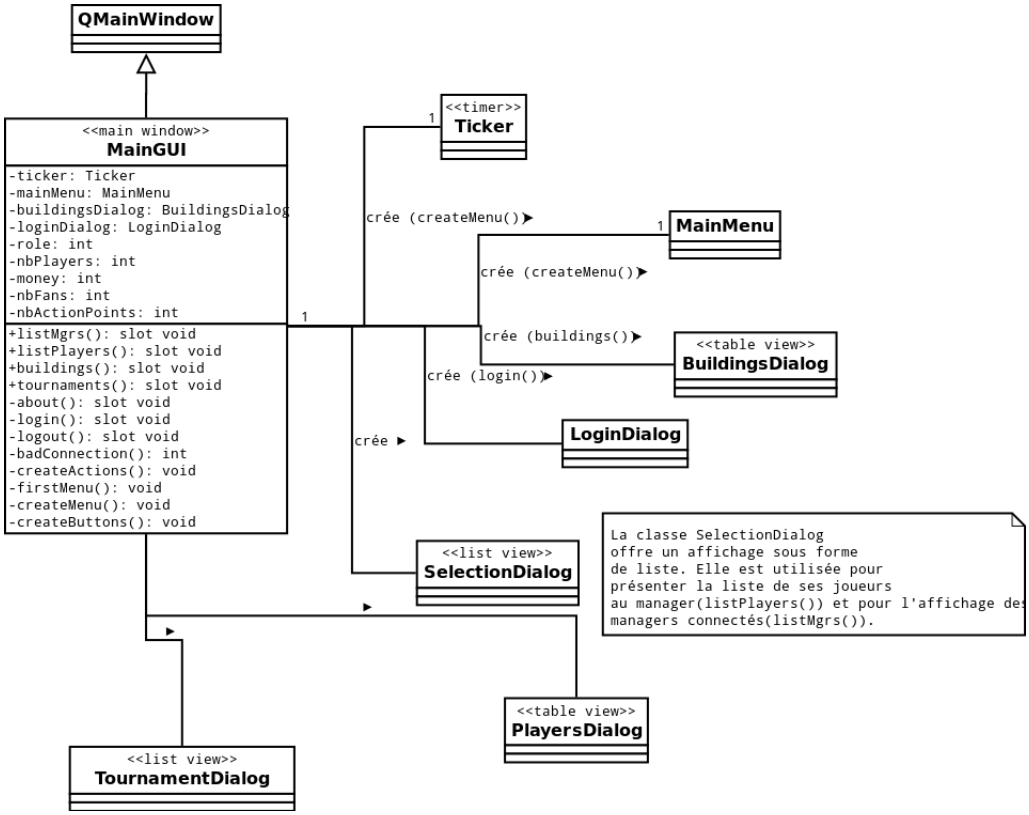

FIGURE 3.20: Diagramme de classes autour de "MainGui"

3.4.3 La bannière

Enfin, le manager voit s'afficher une bannière qui reprend les informations concernant ses avoirs ; ces informations sont rafraîchies périodiquement *(classe "ticker", qui, lorsqu'elle est affichée, lance un timer[1])*. L'utilisateur attentif remarquera que cette bannière se déplace : ce n'est pas un gadget ; à chaque déplacement, les données sont mises à jour ! *(le but est, pour les développeurs, de pouvoir vérifier ainsi aisément que le timer fonctionne).*
Dans certaines parties du jeu, la bannière, moins utile, est cachée à dessein.

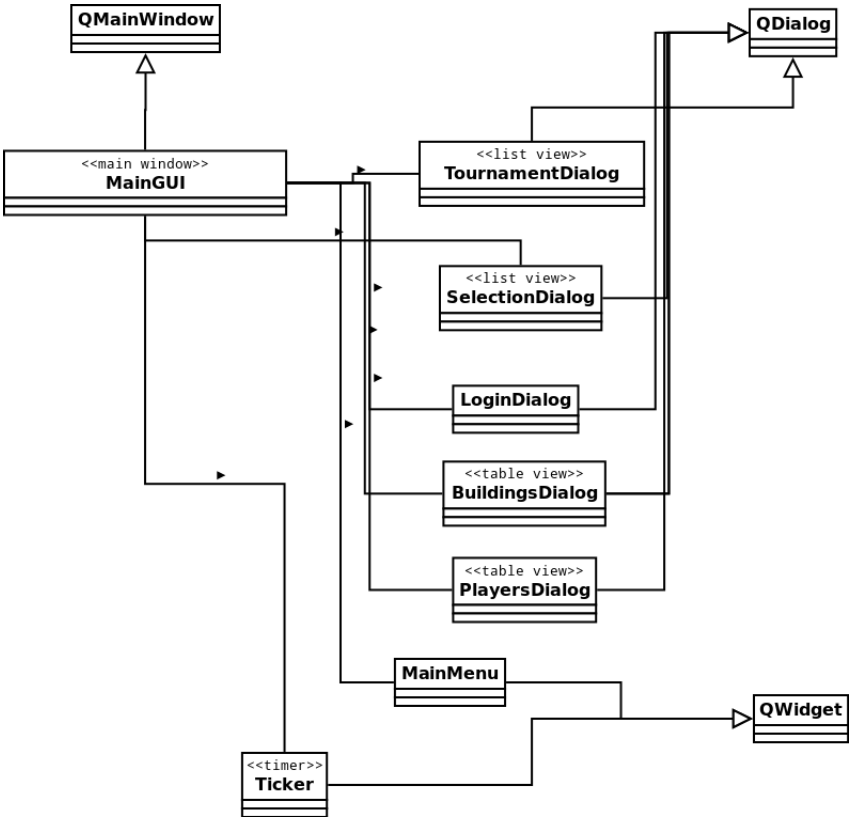

FIGURE 3.21: Diagramme de classes GUI

Comme le montre la figure 3.21, la majorité des classes graphiques héritent de "QDialog", qui elle-même hérite de "QWidget", tout comme "QMainWindow".

3.4.4 La gestion des bâtiments

Sous la forme d'un tableau de bord rafraîchi périodiquement *(classe "BuildingsDialog")*, chaque ligne reprend l'état d'un bâtiment *(une construction model-viewer où le modèle "BuildingsModel", héritant de la classe abstraite "QAbstractTableModel", a été en partie réimplémenté ; un "reset()" avertit le viewer standard "QTableView" lorsque les données sont rafraîchies ; le modèle a été conçu pour être réutilisé, ce qui est le cas dans la gestion des joueurs : figure 3.22).*

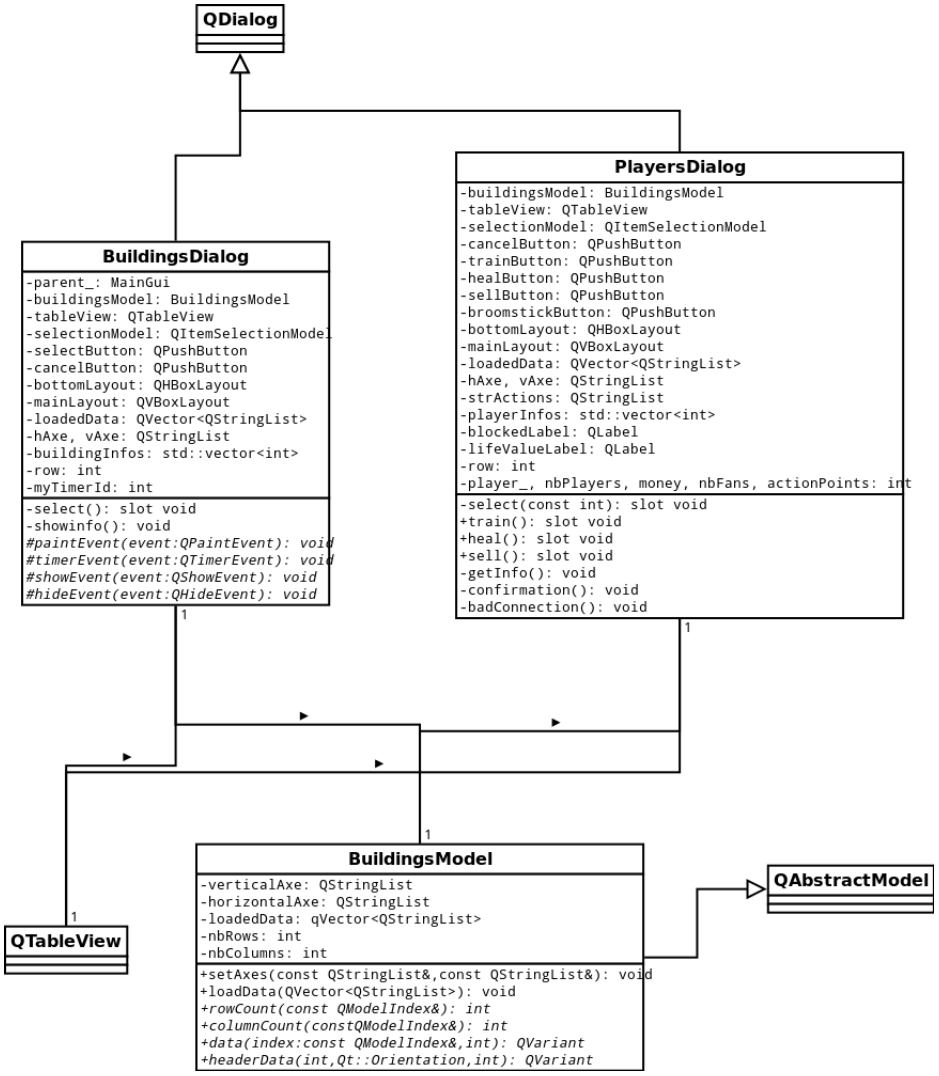

FIGURE 3.22: Diagramme des classes utilisant le modèle "BuildingsModel" associé au viewer "QTableView"

Le manager a la possibilité de demander un upgrade du bâtiment sur la ligne duquel il a placé le curseur, si, bien entendu, il possède assez d'argent et de points d'action, et que le bâtiment n'est pas déjà en cours d'upgrade. Le manager peut entreprendre, à ces conditions, l'upgrade de plusieurs bâtiments. S'il a la patience d'attendre la fin de ceux-ci, il verra, sans devoir intervenir, les données du tableau de bord s'actualiser.

3.4.5 La gestion des joueurs

Cette fonction *("playerMgr()")* commence par la présentation d'une liste *(une construction model-viewer autour d'une liste, placée dans une classe générique "SelectionDialog",*

qui est réutilisée dans plusieurs autres fonctions) dans laquelle le manager peut sélectionner un joueur. Cette présentation est un standard de l'application, repris dans toutes les fonctions ; elle permet de limiter au maximum le nombre d'entrées dans les menus, car une fois le joueur sélectionné, le programme lui présente les informations détaillées concernant ce dernier et propose diverses actions sous forme de boutons, en fonction des caractéristiques du joueur sélectionné. Nous appellerons par la suite ce dispositif un modèle liste-détails-actions.
Les informations détaillées concernant le joueur sont ici à nouveau présentées sous la forme d'un tableau de bord et d'une bannière *(QLabel)*, en légende cette fois. Si le joueur n'est pas bloqué car en cours d'entraînement, de soins ou de vente, et que le manager dispose d'action points, ce dernier voit apparaître des boutons lui permettant de lancer un entraînement, une vente ou d'envoyer le joueur se faire soigner. Bien entendu, ces boutons ne sont affichés que s'ils sont pertinents par rapport à la situation présente du joueur.
Comme le manager ne peut entreprendre qu'une action à la fois, le joueur étant alors bloqué pour un certain temps, la fonction s'arrête ensuite *(Ce dialogue est géré par la classe "playersDialog", qui, pour le tableau de bord, met à nouveau en œuvre une construction model-viewer autour de la classe "BuildingsModel")*. Notons que l'achat d'un broomstick est prévu mais pas encore programmé.

3.4.6 Les tournois

Cette fonction *("tournaments()")* commence par un avertissement ; ensuite, comme pour la gestion des joueurs, vient la présentation d'une liste *(une construction model-viewer gérée par la classe "SelectionDialog")* dans laquelle le manager peut sélectionner un tournoi pour y participer, si la liste n'est pas vide ; sinon, il ne peut que quitter le dialogue.
Par contre, si l'utilisateur est un administrateur, il peut créer un nouveau tournoi *(Comme le nombre de données à introduire est limité, nous avons eu recours ici, comme pour la vente d'un joueur, à une suite de questions, plutôt qu'à la construction d'une classe adhoc héritant de "QDialog")*.

Bibliographie

[1] Blanchette Jasmin, Summerfield Mark - Trolltech 2006 - C++ GUI programming with Qt4 - ISBN 0-13-187249-4

[2] Bulfone Christian - 2012 - Le modèle client-serveur - www.gipsa-lab.fr/ christian.bulfone/IC2A-DCISS