SOFTWARE REQUIREMENTS DOCUMENT

Quidditch Manager 2014

Groupe 4 : Manon Legrand, Hélène Plisnier, Audry Delestree, David Bertha
INFOF209
Université Libre de Bruxelles
Version 1.0

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 par 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 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

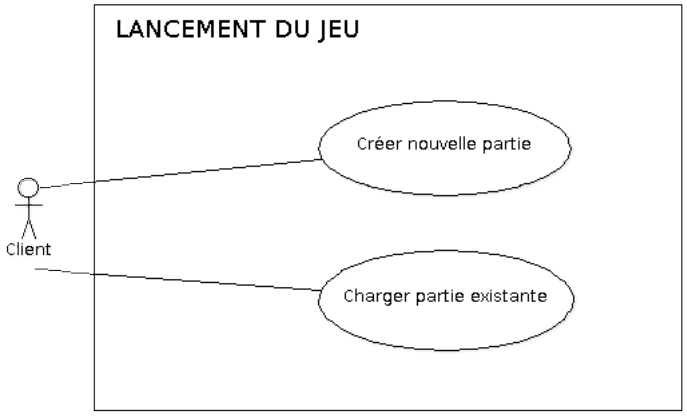

FIGURE 2.1: Cas d'utilisation 1 : Démarrer le jeu

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 et lui demande d'entrer un autre nom. Ceci termine ce use case.

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** :
  - Il n'y a pas de partie existante. Le programme le signale au client et lui propose de se créer une nouvelle partie. Ceci termine ce use case.
  - L'identification de la partie a échoué (mauvais nom ou mot de passe). Le programme le signale au client et lui redemande d'entrer le nom de la partie ainsi que le mot de passe. Ceci termine ce use case.

2.1.2 Cas d'utilisation 2 : jouer un match

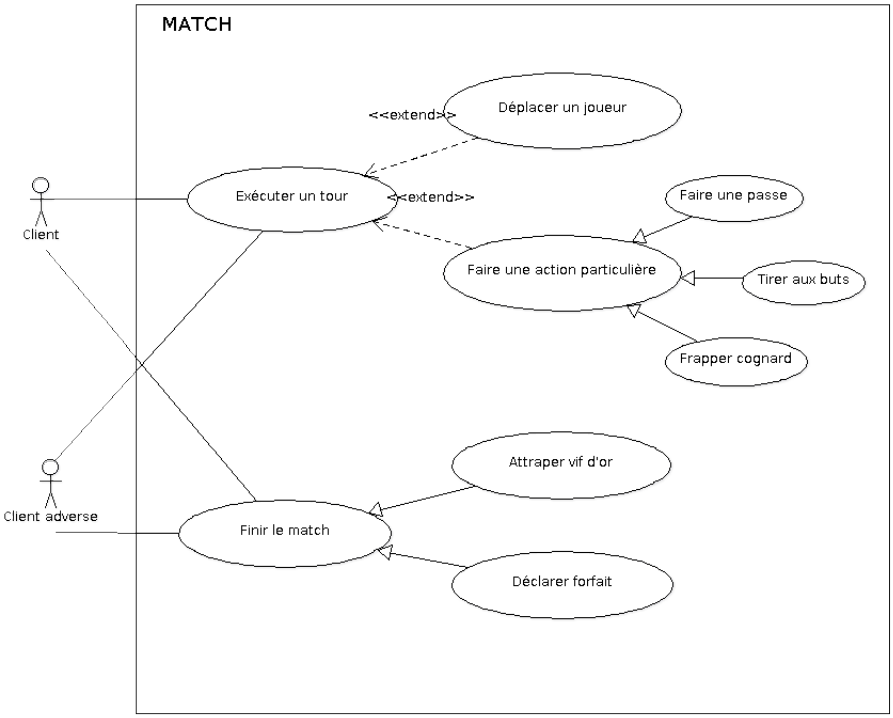

FIGURE 2.2: Cas d'utilisation 2 : Jouer un match

Tous les X temps, le Manager est invité à affronter un Manager adverse dans un match soit à domicile, soit à l'extérieur (cette information déterminera à qui reviendront les recettes générées par la vente de tickets). Avant de commencer le match proprement dit, le Manager doit choisir sept de ses joueurs pour former son équipe. Une fois le match terminé, le Manager prend connaissance du résultat du match (s'il a gagné ou perdu) et des recettes éventuellement réalisées. Un match peut être amical (démarré à la suite d'une invitation pouvant être envoyée à tout moment par un Manager à un autre Manager connecté) ou s'inscrivant dans un championnat. Les deux types de match ont pour point commun de rendre le Manager gagnant plus riche d'au moins une prime de victoire.

Exécuter un tour

- **Relations avec d'autres cas d'utilisation** : Est étendu par Déplacer un joueur et Faire une action particulière
- **Pré-conditions** : L'action se déroule lors d'un match.
- **Post-conditions** : Un tour a été effectué et chaque joueur sur le terrain a (éventuellement) bougé.
- **Cas général** : Un match se déroule tour par tour. À chaque tour, les clients qui s'affrontent doivent indiquer, pour chaque joueur, un mouvement à effectuer ou simplement les laisser sur place.
- **Cas exceptionnels** :
  - L'adversaire s'est déconnecté et ne s'est pas reconnecté dans le temps imparti. Le programme signale la déconnexion inopinée et le match s'arrête.

Déplacer un joueur

- **Relations avec d'autres cas d'utilisation** : Etend Exécuter un tour
- **Pré-conditions** : Le joueur sélectionné pour faire un déplacement n'a pas été choisi pour une action particulière.
- **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 Manager 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** : Etend Exécuter un tour. Généralise Faire une passe, Tirer aux buts et Frapper cognard
- **Pré-conditions** : Aucune action particulière n'a encore été assignée au joueur sélectionné.
- **Post-conditions** : Le joueur a une action particulière à accomplir.
- **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 .

- **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. Attraper le souafle suite à une passe ou l'intercepter (passe adverse) se fait de façon automatique par un poursuiveur se trouvant sur la trajectoire de la passe.
- **Cas exceptionnels** :
  - Le joueur est heurté par un cognard. La passe échoue.

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. Intercepter le souafle se fait automatiquement par le gardien ou les poursuiveurs adverses s'ils se trouvent sur la trajectoire du souafle. Si le tir est réussi, l'équipe qui vient de marquer gagne 10 points.
- **Cas exceptionnels** :
  - Le poursuiveur qui s'apprête à tirer est heurté par un cognard. Le tir au but échoue.

Frapper 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 client indique dans quelle direction le batteur doit frapper le cognard.
- **Cas exceptionnels** :
  - Le batteur rate son coup. Le cognard heurte le joueur.

Finir le match

- **Relations avec d'autres cas d'utilisation** : Généralise Attraper vif d'Or et Déclarer forfait
- **Pré-conditions** : Cette action se déroule lors d'un match.
- **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, popularité) chaque manager.

- **Cas exceptionnels** :
  - Le match se finit parce que les deux clients qui s'affrontent se sont déconnectés et n'ont pas joué le match jusqu'au bout. Le match est considéré comme annulé.
  - Si les 7 joueurs d'une même équipe sont blessés et ne peuvent continuer à jouer, l'équipe adverse gagnera le match. Les joueurs blessés sont envoyés à l'infirmerie et sont bloqués jusqu'à leur rétablissement.

Attraper vif d'or

- **Relations avec d'autres cas d'utilisation** : Spécialise 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 a repéré le vif d'or, c'est-à-dire qu'en se déplaçant sur le terrain, il est arrivé suffisamment proche du vif d'or pour que sa précision permette de le repérer (et de le rendre visible pour son Manager). L'attrapeur essaiera automatiquement 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 est heurté par un cognard. Le vif d'or reste libre.

Déclarer forfait

- **Relations avec d'autres cas d'utilisation** : Spécialise Finir match
- **Pré-conditions** : L'action se déroule au cours d'un match.
- **Post-conditions** : Le match est terminé. Le Manager qui n'a pas déclaré forfait a gagné.
- **Cas général** : Un des Managers 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 Manager se déconnecte au cours d'un match et ne se reconnecte pas dans un certain délai de temps, on considérera qu'il a déclaré forfait.

2.1.3 Cas d'utilisation 3 : gérer ses bâtiments

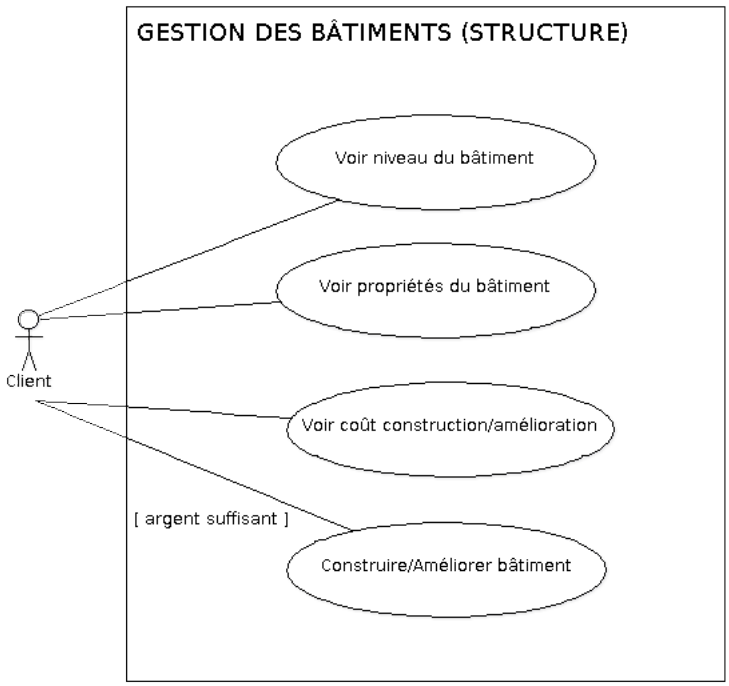

FIGURE 2.3: Cas d'utilisation 3 : Gérer ses bâtiments

Différents bâtiments sont disponibles à la construction pour le Manager, moyennant un certain coût et un temps d'attente (avant fin des travaux). Généralement : une fois construits, le Manager peut améliorer ses bâtiments et accéder aux informations les concernant. Certains bâtiments offrent des usages supplémentaires.

Voir niveau du bâtiment

- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : On a sélectionné un bâtiment.
- **Post-conditions** : Néant.
- **Cas général** : Le client souhaite savoir quel est le niveau actuel (de construction et d'amélioration) d'un certain bâtiment.

- **Cas exceptionnels** : Néant.

Voir propriétés du bâtiment

- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : On a sélectionné un bâtiment.
- **Post-conditions** : Néant.
- **Cas général** : Le client souhaite voir les propriétés d'un certain bâtiment (ces propriétés dépendent du niveau et du bâtiment).
- **Cas exceptionnels** :
  - Le bâtiment est de niveau 0, il n'a pas encore été construit et n'a donc actuellement pas de propriétés effectives.

Voir coût construction/amélioration

- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : On a sélectionné un bâtiment.
- **Post-conditions** : Néant.
- **Cas général** : Le client souhaite connaître la somme requise pour pouvoir lancer un projet de construction ou d'amélioration d'un certain bâtiment.
- **Cas exceptionnels** : Néant.

Construire/Améliorer bâtiment

- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Le Manager possède suffisamment d'argent pour payer le projet de construction ou d'amélioration. Le Manager a choisi le bâtiment qu'il désire améliorer.
- **Post-conditions** : Le projet sera terminé après un certain laps de temps déterminé par le niveau du bâtiment. La date de fin des travaux est sauvegardée dans un calendrier qui sera vérifié régulièrement.
- **Cas général** : Le Manager souhaite construire (si niveau actuel vaut 0) ou améliorer (si niveau actuel vaut 1 ou plus) un certain bâtiment et choisit donc de lancer un projet de construction/amélioration. Le coût des travaux est déduit de son capital et les travaux ne sont achevés qu'après un certain temps.
- **Cas exceptionnels** :
  - Le bâtiment sélectionné est à son niveau maximum. Aucune évolution supplémentaire ne peut être effectuée sur ce bâtiment.

2.1.4 Cas d'utilisation 4 : gérer son équipe

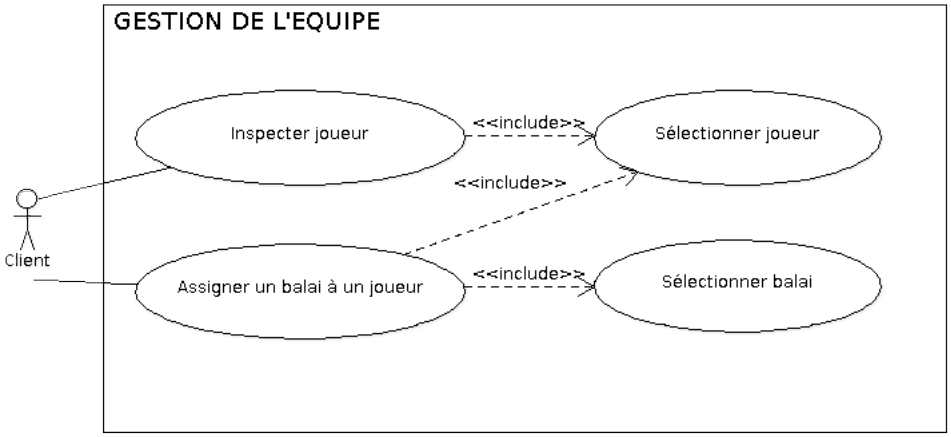

FIGURE 2.4: Cas d'utilisation 4 : gérer son équipe

Le manager a une vue sur tous les joueurs de son club; leur description et leurs caractéristiques. Il lui est également donné la possibilité d'assigner un nouveau balai à tel ou tel joueur.

Inspecter joueur

- **Relations avec d'autres cas d'utilisation** : Inclut Sélectionner joueur
- **Pré-conditions** : Le Manager a sélectionné un de ses joueurs.
- **Post-conditions** : Néant.
- **Cas général** : Le client souhaite se renseigner sur les capacités d'un de ses joueurs. Il pourra voir le niveau de chaque attribut du joueur ainsi que sa popularité et son balai.
- **Cas exceptionnels** : Néant.

Assigner un balai à un joueur

- **Relations avec d'autres cas d'utilisation** : Inclut Sélectionner joueur et Sélectionner balai
- **Pré-conditions** : Le Manager a sélectionné un balai disponible pour une assignation.
- **Post-conditions** : Le balai sélectionné est assigné au joueur. L'ancien balai est à nouveau disponible pour une assignation.
- **Cas général** : Le Manager souhaite changer le balai d'un joueur. Les balais ont des bonus particuliers qui permettent d'augmenter certains attributs des joueurs.

Une fois u'un joueur s'est vu assigné un nouveau balai, l'ancien n'est plus assigné à aucun joueur mais reste disponible.
- **Cas exceptionnels** :
  - Le Manager retire le balai d'un joueur sans lui en assigner un nouveau s'il n'a pas au préalable sélectionné de balai disponible.

Sélectionner joueur

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Inspecter joueur et Assigner un balai à un joueur
- **Pré-conditions** : Le Manager a une vue sur tous ses joueurs.
- **Post-conditions** : Le Manager a accès aux caractéristiques propres au joueur sélectionné.
- **Cas général** : Le client choisit le joueur qu'il veut gérer (inspecter ou changer le balai).
- **Cas exceptionnels** : Néant.

Sélectionner balai

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Assigner un balai à un joueur
- **Pré-conditions** : Le balai doit être disponible, c'est-à-dire qu'il ne doit être assigné à aucun joueur.
- **Post-conditions** : Le balai disponible est sélectionné.
- **Cas général** : Le client choisit le balai qu'il va assigner à un joueur. Une fois assigné, le balai ne sera plus disponible.
- **Cas exceptionnels** : Néant.

2.1.5 Cas d'utilisation 5 : magasins

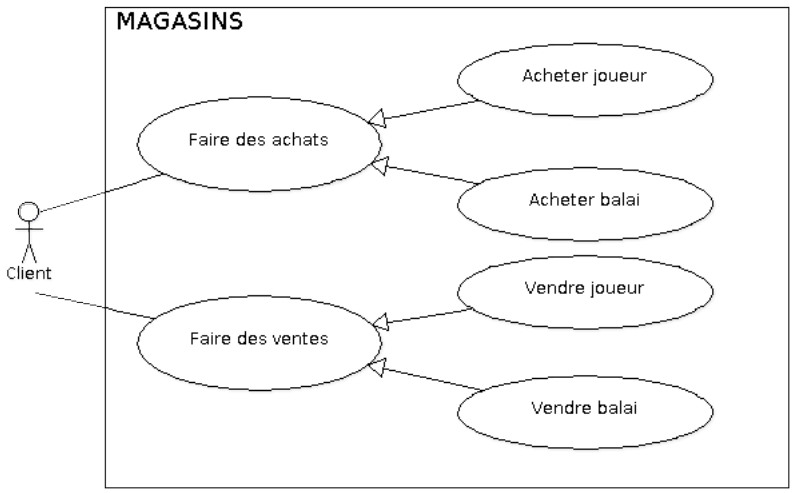

FIGURE 2.5: Cas d'utilisation 5 : Magasins

Dans le bâtiment « centre de recrutement » , le Manager pourra "s'offrir" de nouveaux joueurs, selon ses moyens. Il peut également vendre les joueurs qu'il possède. Les joueurs sont vendus aux enchères. Le vendeur décide du prix de vente dans une fourchette de prix (estimée d'après les capacités du joueur). Dans le bâtiment « magasin de balais » , le Manager pourra acheter ou vendre des balais. Outre les balais de base qui permettent juste de voler, il y a aussi des balais qui octroient des bonus à certaines capacités des joueurs.

Faire des achats

- **Relations avec d'autres cas d'utilisation** : Généralise Acheter joueur et Acheter balai
- **Pré-conditions** : Le Manager doit posséder suffisamment d'argent pour payer ce qu'il veut acheter. Le Manager doit avoir de la place pour un joueur ou pour un balai.
- **Post-conditions** : Le prix est déduit du solde du manager. L'item se trouve en la possession du Manager.
- **Cas général** : Le client souhaite acheter un nouveau joueur pour renforcer son équipe ou un nouveau balai pour renforcer un joueur.
- **Cas exceptionnels** : Néant.

Faire des ventes

- **Relations avec d'autres cas d'utilisation** : Généralise Vendre joueur et Vendre balai.
- **Pré-conditions** :
  - Le Manager a choisit le joueur ou le balai qu'il désire vendre.
  - L'item que le Manager désire vendre doit être disponible. S'il s'agit d'un balai, il ne peut être assigné à un joueur lors de sa mise en vente. S'il s'agit d'un joueur, il doit être libre (non bloqué par une blessure, un entraînement ou une tournée de promotion).
- **Post-conditions** : Le Manager reçoit l'argent de la vente. Il ne possède plus l'item vendu. L'item devient disponible à l'achat.
- **Cas général** : Le Manager souhaite vendre un joueur ou un balai s'il n'en a plus besoin ou si ses finances vont mal.
- **Cas exceptionnels** : Néant.

Acheter joueur

- **Relations avec d'autres cas d'utilisation** : Spécialise Faire des achats
- **Pré-conditions** : Le Manager s'est manifesté en sélectionnant le joueur qui l'intéresse dans la liste des joueurs mis en vente.
- **Post-conditions** : S'il est le seul enchérisseur, le Manager remporte l'enchère. S'ils sont plusieurs à enchérir, le Manager peut participer au tour suivant. Si personne n'enchérit au tour suivant et que le Manager était le dernier enchérisseur du tour courant, il remporte l'enchère. Sinon, le Manager n'a pas remporté l'enchère.
- **Cas général** : Le client choisit le joueur qu'il veut acheter selon les capacités de ce dernier (ce qui l'intéresse), et selon le prix de ce joueur. Les joueurs achetés sont équipés d'un balai médiocre qui n'a pas de valeur financière. Les enchères se déroulent par tours. Le Manager manifeste son intérêt en se signalant au premier tour. Il assure ainsi sa participation au tour suivant. Si le prix du joueur devient trop élevé à son goût, le Manager peut abandonner en ne renouvellant pas son enchère lors d'un tour, ce qui lui interdira de surenchérir lors du tour suivant.
- **Cas exceptionnels** :
  - Personne ne se manifeste au premier tour. La vente est annulée, le joueur retourne en la possession du vendeur.
  - Un Manager souhaite surenchérir alors que le prix, augmenté de son enchère, est au-dessus de ses moyens. Le programme le lui signale et son offre n'est pas prise en compte.
  - Un Manager désire surenchérir alors qu'il n'a pas émis d'enchère au tour précédant. Le programme le lui signale et ne considère pas son offre.

Acheter balai

- **Relations avec d'autres cas d'utilisation** : Spécialise Faire des achats
- **Pré-conditions** : Néant.

- **Post-conditions** : Néant.
- **Cas général** : Le client choisit le balai qu'il veut acheter en fonction des caractéristiques de celui-ci et de son prix.
- **Cas exceptionnels** :
  - Des balais médiocres sans bonus sont disponibles gratuitement.

Vendre joueur

- **Relations avec d'autres cas d'utilisation** : Spécialise Faire des ventes
- **Pré-conditions** : Néant.
- **Post-conditions** : Néant.
- **Cas général** : Le client choisit le joueur qu'il veut vendre et à quel prix (entre des bornes supérieure et inférieure imposées). Le joueur ne sera plus dans l'équipe mais sera disponible à l'achat. Le client peut vendre aux enchères un joueur équipé d'un balai, ce qui augmente un peu la valeur du joueur.
- **Cas exceptionnels** :
  - Le joueur n'a pas trouvé d'acheteur. Il reste dans l'équipe du Manager/vendeur.
  - Le client n'a que 7 joueurs dans son équipe et en vend un. Il ne lui reste pas suffisamment de joueurs pour avoir une équipe complète, il recevra donc un joueur médiocre avec des attributs et un balai de base.

Vendre balai

- **Relations avec d'autres cas d'utilisation** : Spécialise Faire des ventes
- **Pré-conditions** : Néant.
- **Post-conditions** : Néant.
- **Cas général** : Le client désire vendre un balai.
- **Cas exceptionnels** :
  - Le Manager se débarrasse de ses balais de base lorsqu'il n'en a plus besoin. Cette vente ne lui rapporte rien.

2.1.6 Cas d'utilisation 6 : améliorer ses joueurs

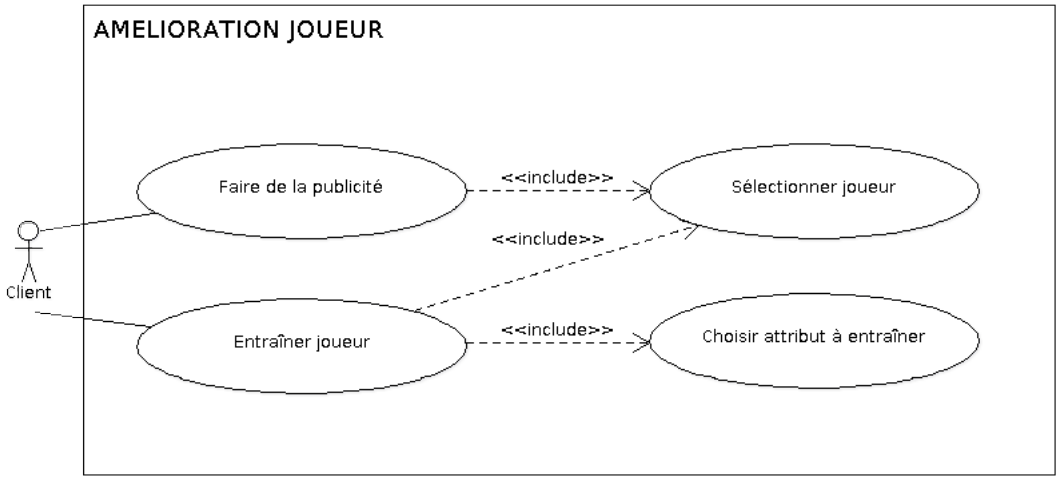

FIGURE 2.6: Cas d'utilisation 6 : Amélioration joueur

Lorsqu'il voudra augmenter les capacités de l'un de ses joueurs, le Manager l'enverra au « centre d'entrainement » . Une séance d'entraînement a un certain coût. Ce coût varie en fonction du niveau d'entraînement déjà atteint par le joueur et en fonction du niveau du bâtiment. Le manager peut également améliorer la popularité d'un joueur en l'envoyant à l'« agence de publicité » . Encore une fois, le Manager doit payer pour bénéficier de ces services. Ces actions rendent le joueur indisponible jusqu'à la fin du temps requis pour réaliser l'entraînement ou la publicité.

Faire de la publicité

- **Relations avec d'autres cas d'utilisation** : Inclut Sélectionner joueur
- **Pré-conditions** : Le Manager dispose de suffisamment d'argent pour payer la publicité.
- **Post-conditions** : Le joueur est bloqué jusqu'à une certaine date sauvegardée dans un calendrier. Ce calendrier sera vérifié régulièrement.
- **Cas général** : Le Manager souhaite augmenter la popularité d'un joueur. Ceci est possible grâce au bâtiment « Agence de publicité » dont le niveau détermine la durée. En payant une certaine somme, le client peut donc augmenter la popularité du joueur choisi. Le joueur est indisponible jusqu'à la fin de la publicité.

- **Cas exceptionnels** :
  - Le Manager n'a pas les moyens de payer la publicité. Le programme le lui signale et l'action est annulée.

Entraîner joueur

- **Relations avec d'autres cas d'utilisation** : Inclut Sélectionner joueur et Choisir un attribut à entraîner.
- **Pré-conditions** : Le client dispose de suffisamment d'argent pour payer l'entraînement.
- **Post-conditions** : Le joueur est bloqué jusqu'à une certaine date sauvegardée dans un calendrier. Ce calendrier sera vérifié régulièrement.
- **Cas général** : Le client souhaite augmenter un attribut d'un joueur. Le niveau du bâtiment concerné par cette action détermine la durée d'indisponibilité du joueur. En payant une certaine somme, le client peut donc augmenter une caractéristique du joueur choisi. En attendant la fin de l'entraînement, le joueur est donc indisponible.
- **Cas exceptionnels** :
  - Le Manager n'a pas les moyens de payer la séance d'entraînement. Le programme le lui signale et l'action est annulée.

Sélectionner joueur

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Faire de la publicité et Entraîner joueur
- **Pré-conditions** : Le joueur sélectionné doit être disponible; c'est-à-dire qu'il ne peut être blessé (car il se trouverait alors à l'infirmerie), ni en train de faire un autre entraînement ou parti en campagne de promotion.
- **Post-conditions** : Le joueur sur lequel on va faire l'amélioration sera indisponible jusqu'à la fin de l'entraînement ou de la publicité.
- **Cas général** : Le Manager choisit de quel joueur il veut augmenter un attribut ou la popularité.
- **Cas exceptionnels** :
  - Le joueur que le Manager désire sélectionner est momentanément indisponible (car en pleine séance d'entraînement, en campagne de promotion où est soigné à l'infirmerie). Le programme le lui signale et l'action est annulée.

Choisir attribut à entraîner

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Entraîner joueur
- **Pré-conditions** : Le Manager a sélectionné un joueur et a décidé de l'entraîner.
- **Post-conditions** : L'attribut choisi sera augmenté lorsque la séance d'entraînement aura pris fin.
- **Cas général** : Le client choisit quel attribut il souhaite augmenter. Il ne peut entraîner qu'un attribut à la fois.
- **Cas exceptionnels** :

– L'attribut choisi est déjà à son maximum. Le programme le lui signale et l'action est annulée.

2.1.7 Cas d'utilisation 7 : le stade

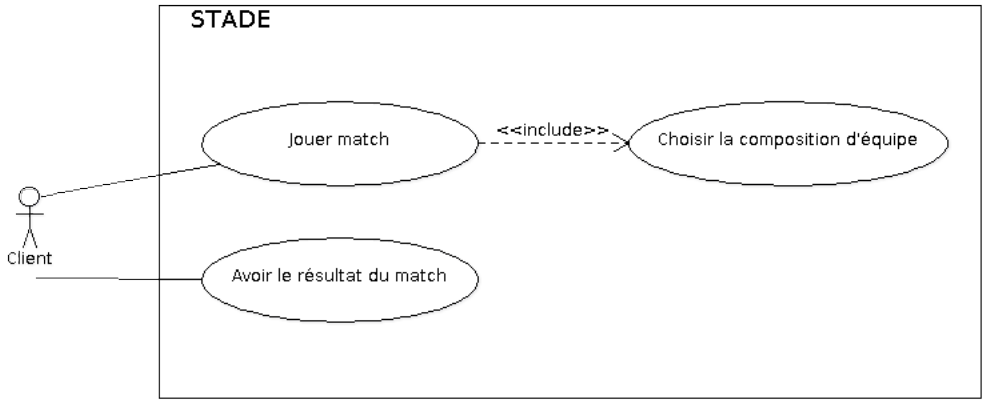

FIGURE 2.7: Cas d'utilisation 7 : Stade

C'est via le stade que le Manager peut lancer un match lorsque c'est un jour de match. Il devra alors choisir les joueurs qui participeront à la rencontre. Une fois le match terminé, il pourra voir les résultats de ce dernier, ainsi que ce qu'il a gagné, ou perdu (argent et popularité).

Jouer match

- **Relations avec d'autres cas d'utilisation** : Inclut Choisir la composition d'équipe
- **Pré-conditions** : C'est un jour de match et l'adversaire est également connecté.
- **Post-conditions** : Il y a un gagnant et un perdant.
- **Cas général** : Pour savoir si c'est un jour de match, le programmme vérifie en checkant le calendrier des matches. S'il y a bien un match de prévu, le Manager est connecté en même temps que son adversaire du jour. Ils doivent donc jouer le match dont la date a été prédéterminée.
- **Cas exceptionnels** :
  - L'adversaire ne se connecte jamais. Le Manager jouera contre l'équipe de son adversaire qui suivra une stratégie automatique minimaliste (Intelligence Artificielle).

Choisir la composition d'équipe

- **Relations avec d'autres cas d'utilisation** : Est inclus dans Jouer match
- **Pré-conditions** : Les joueurs choisis doivent être disponibles et être équipés d'un balai.
- **Post-conditions** : Néant.
- **Cas général** : Le client choisit les 7 joueurs qui seront sur le terrain pour le match qu'il doit jouer.
- **Cas exceptionnels** :
  - Le Manager n'en choisi aucun. 7 joueurs sont alors sélectionnés aléatoirement.

Avoir le résultat du match

- **Relations avec d'autres cas d'utilisation** : Néant.
- **Pré-conditions** : Le match doit être terminé.
- **Post-conditions** : Néant.
- **Cas général** : Après le match, le client peut voir le résultat du match et ce qui s'en suit ; l'argent gagné et l'effet sur la popularité des joueurs.
- **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

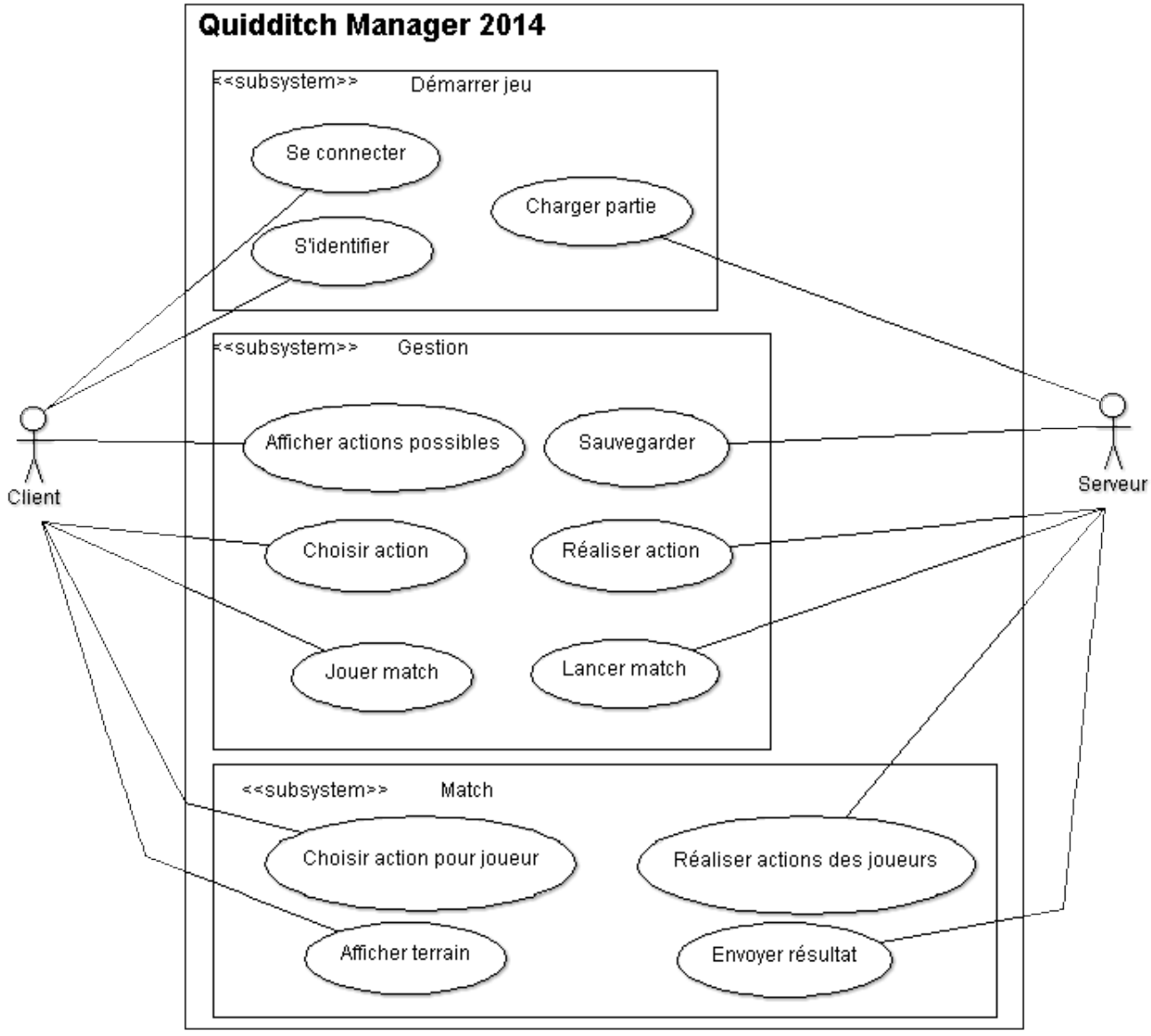

FIGURE 3.1: Cas d'utilisation 1 : Quidditch Manager 2014

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** : Au moins un des clients qui doivent s'affronter lors de ce match est connecté.
- **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** :
  - Un des clients ne s'est pas connecté pour le match, le serveur prendra sa place pour déplacer ses joueurs et affronter le client connecté.
  - Aucun client ne s'est connecté pour le match, le match est considéré comme annulé.

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 souafle 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 Diagrammes de classes

Nous allons ici présenter les différentes classes et leurs relations.

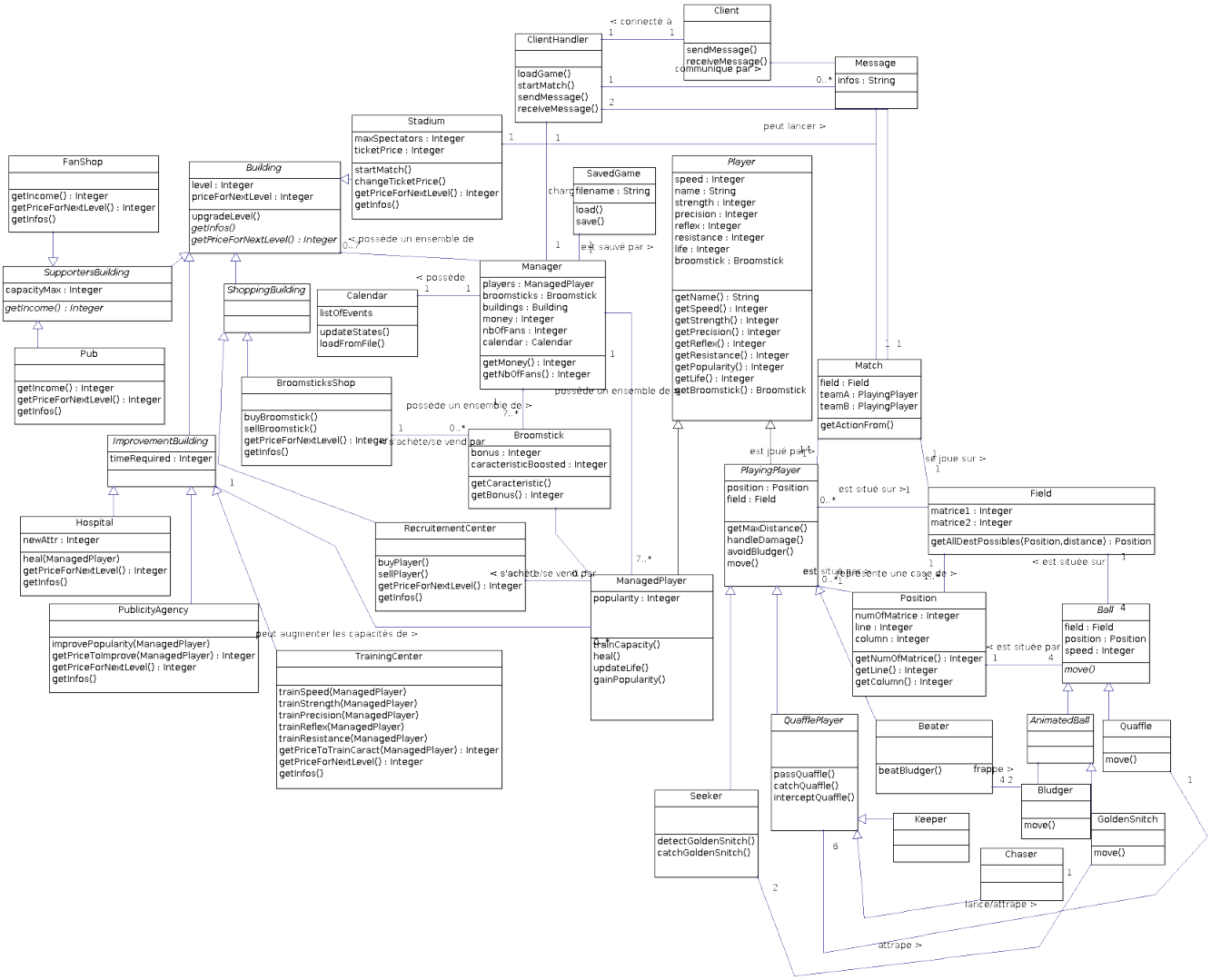

FIGURE 3.2: Diagramme de classe général

Les joueurs

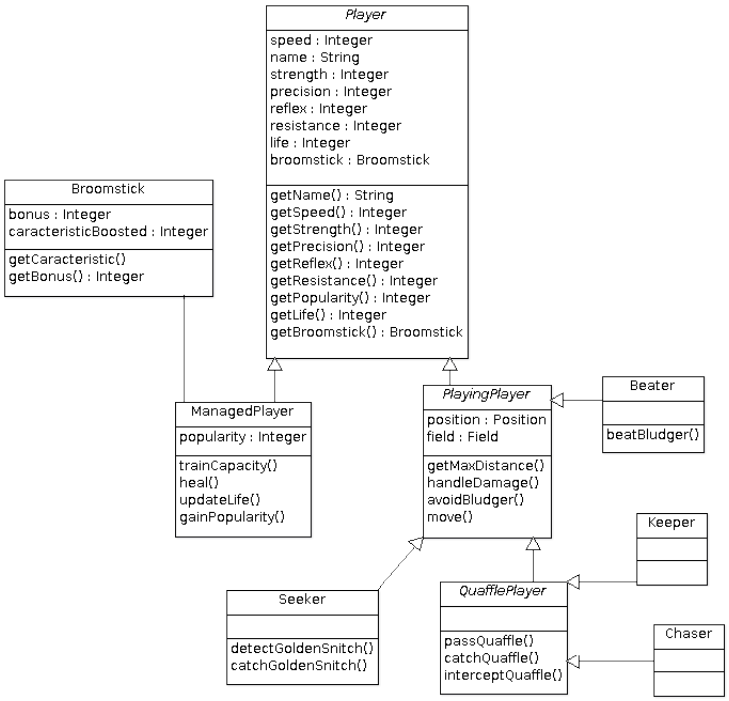

FIGURE 3.3: Diagramme des classes joueurs

Une classe abstraite réprésente le joueur avec ses caractéristiques de base. Dans un soucis de découplage, nous différencions deux classes spécialisées à partir de cette base. Le joueur au niveau de la gestion est distingué du joueur au niveau du match. En effet, ils nécessitent chacun des méthodes fort différentes : d'un côté des méthodes d'amélioration d'attributs, de l'autre des méthodes d'actions spécifiques à un match. Les attributs du joueur de match, ce dernier pouvant se construire depuis un joueur au niveau de la gestion, ne sont pas une simple transposition des caractéristiques du joueur en gestion. En effet, intervient notamment le calcul du bonus lié au balai.

Concernant le réseau

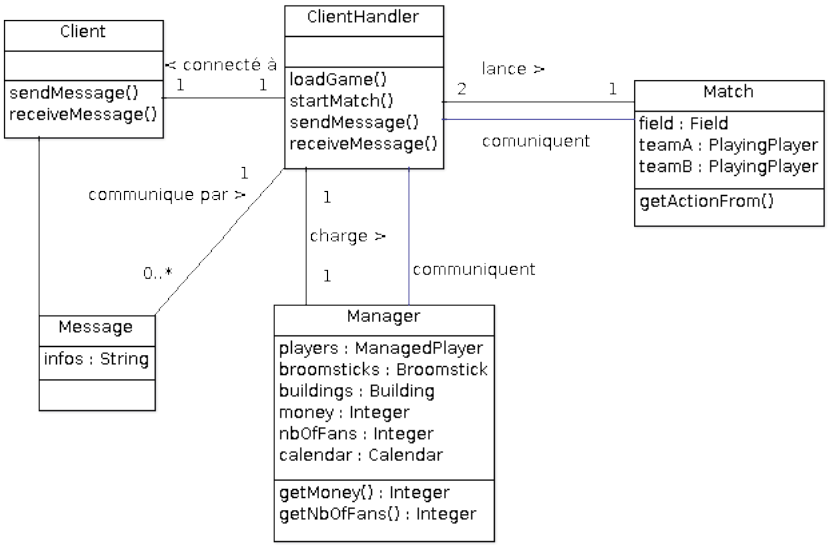

FIGURE 3.4: Diagramme des classes impliquées dans le réseau

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 sera écrite en 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.

Concernant la gestion générale

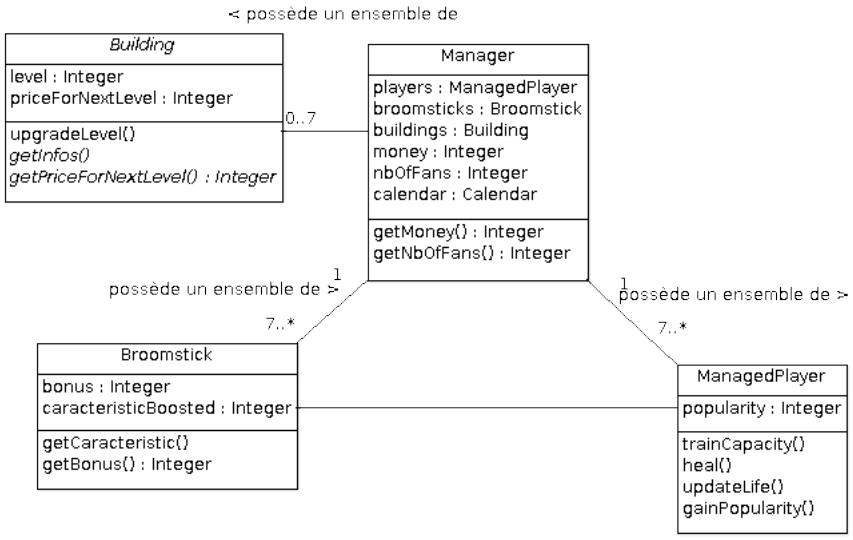

FIGURE 3.5: Diagramme des classes impliquées dans la partie gestion

Le manager est la classe dédiée à la partie gestion. Elle représente une partie du jeu, composée d'un ensemble de joueurs, de balais, de bâtiments, d'un calendrier et d'un trésor. Elle donne accès aux bâtiments qui chacun donne accès aux possibilités d'améliorations, d'achats et de vente.

Concernant le match

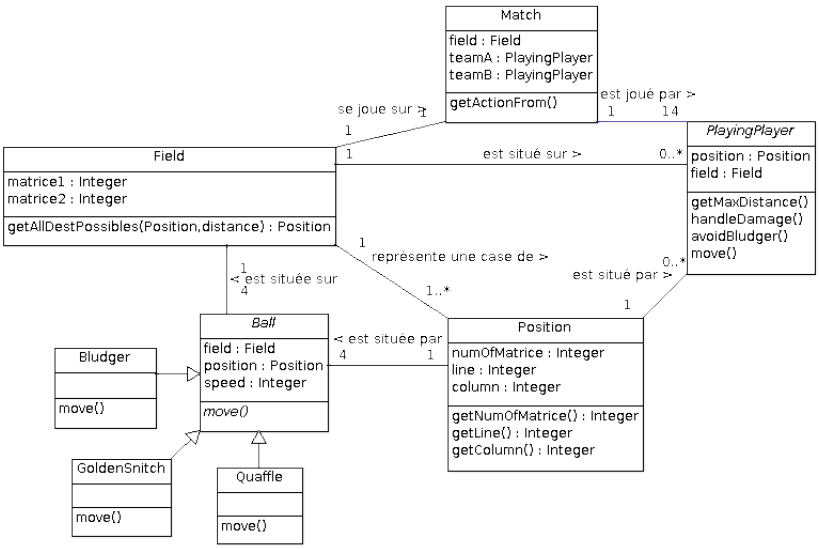

FIGURE 3.6: Diagramme des classes impliquées dans les matchs

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. La classe *Position* regroupe les informations de position d'un joueur ou d'une balle sur le terrain.

Les bâtiments

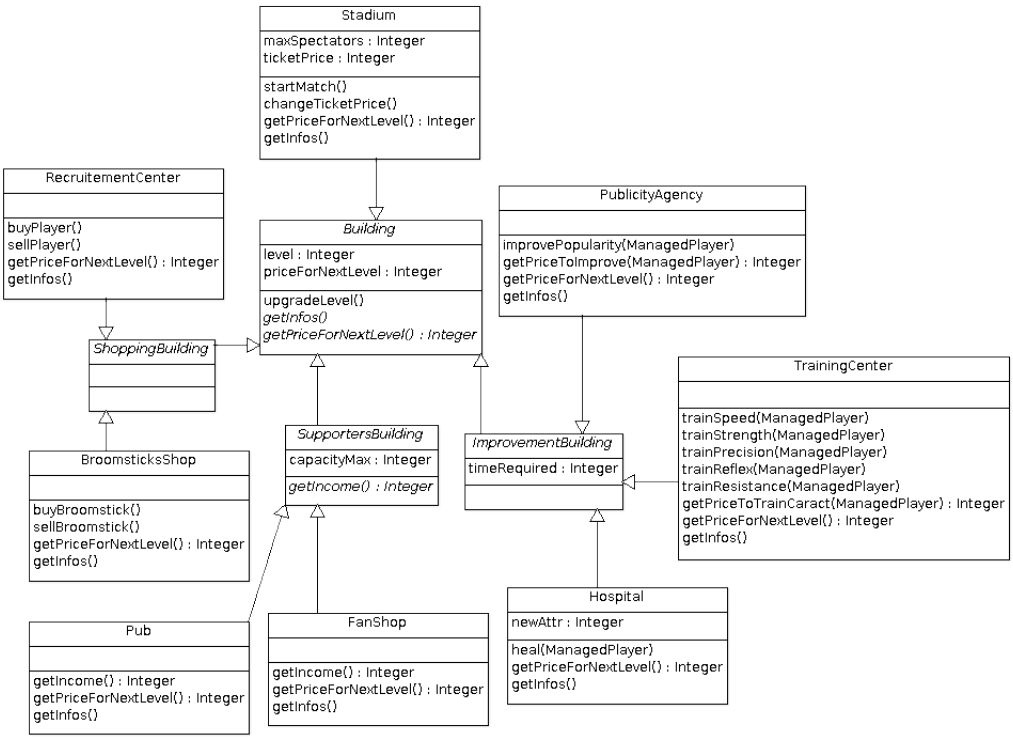

FIGURE 3.7: Diagramme des classes de bâtiments

Une classe abstraite représente tout bâtiment. Cette classe est dérivée dans d'autres classes abstraites qui regroupent les types de bâtiments selon leur rôle principal (achat/vente, augmentation d'une capacité), plus le stade à partir duquel un nouveau match peut démarrer. Par exemple, l'infirmerie fait partie des bâtiments améliorant les joueurs puisqu'elle soigne leurs blessures. Les attributs communs sont le niveau (chaque bâtiment à un niveau, le niveau 0 signifiant « pas encore construit ») ainsi que le prix pour passer au niveau suivant.

Concernant les fichiers

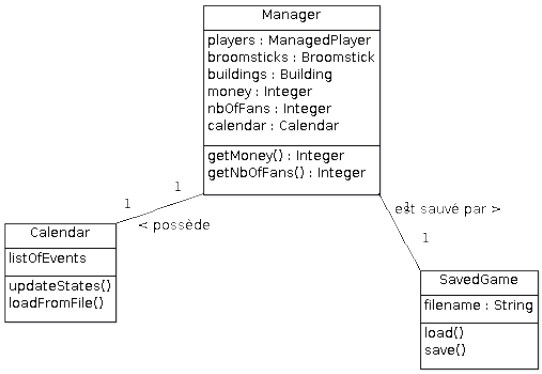

FIGURE 3.8: Diagramme des classes impliquées dans l'enregistrement

L'état d'une partie aussi bien que les actions bloquantes ou planifiées doivent être enregistrées pour perdurer entre deux connexions. Pour cela, nous utilisons deux classes qui vont permettre de charger des données depuis le disque ou de les y sauvegarder.

3.3.2 Diagramme de séquence

Voici un diagramme de séquence qui illustre la gestion des différents rôles du serveur au travers de différents processus.

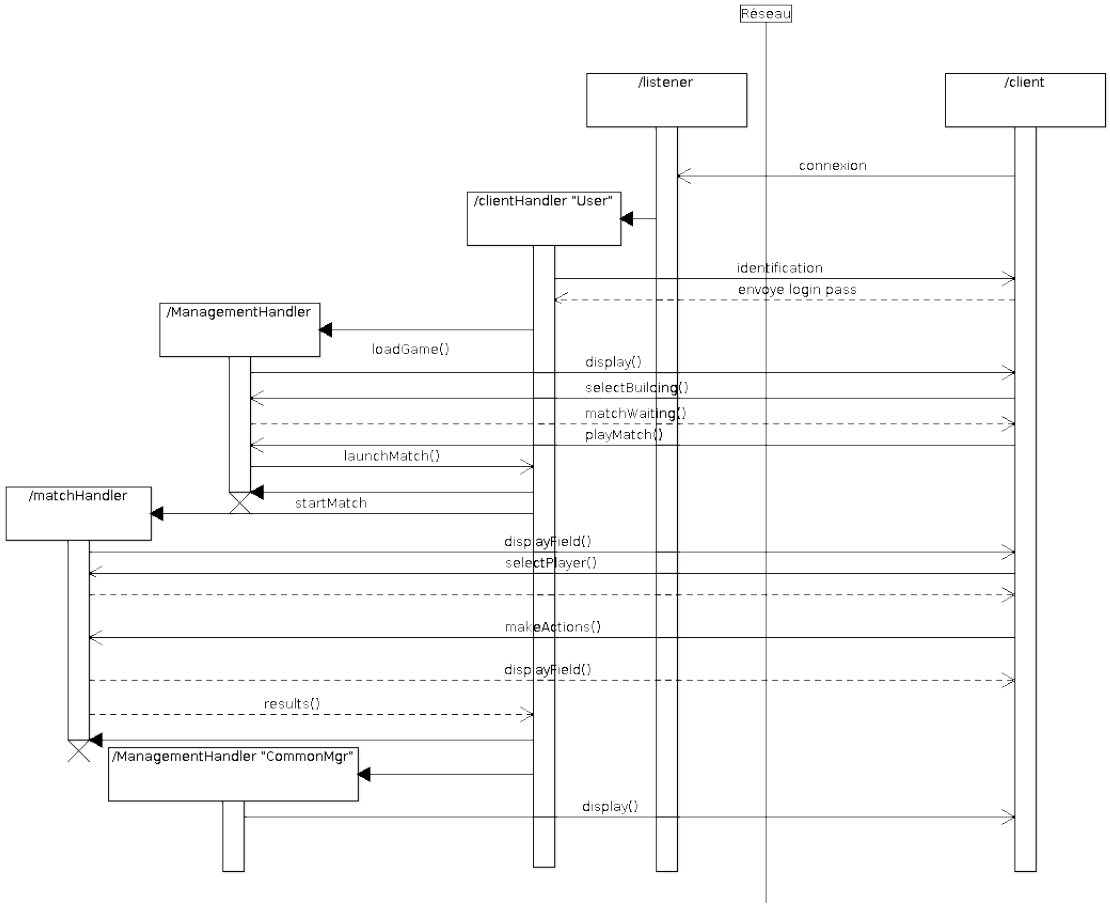

FIGURE 3.9: Diagramme des classes impliquées dans le réseau

Le côté gestion est bien dissocié du côté match, et le *clientHandler* assure la communication avec le client. C'est le *clientHandler* du manager qui reçoit le match dans son stade qui va créer l'instance en charge du match.

3.3.3 Le réseau

L'interface réseau a pour fonction de faire communiquer de façon bidirectionnelle :

- des clients chargés de communiquer chacun avec un manager : 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.

3.3.4 Synopsis du serveur

- Au démarrage, le serveur crée une instance de la classe CommonMgr, destinée à gérer initialisation et clôture des activités fonctionnelles sur le serveur ;
- Le serveur se place 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 figure 3.10) ;
- 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 cmdHandler de l'objet 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) et, dans le futur (interface graphique), devrait lui permettre éventuellement d'entreprendre plusieurs activités en parallèle.

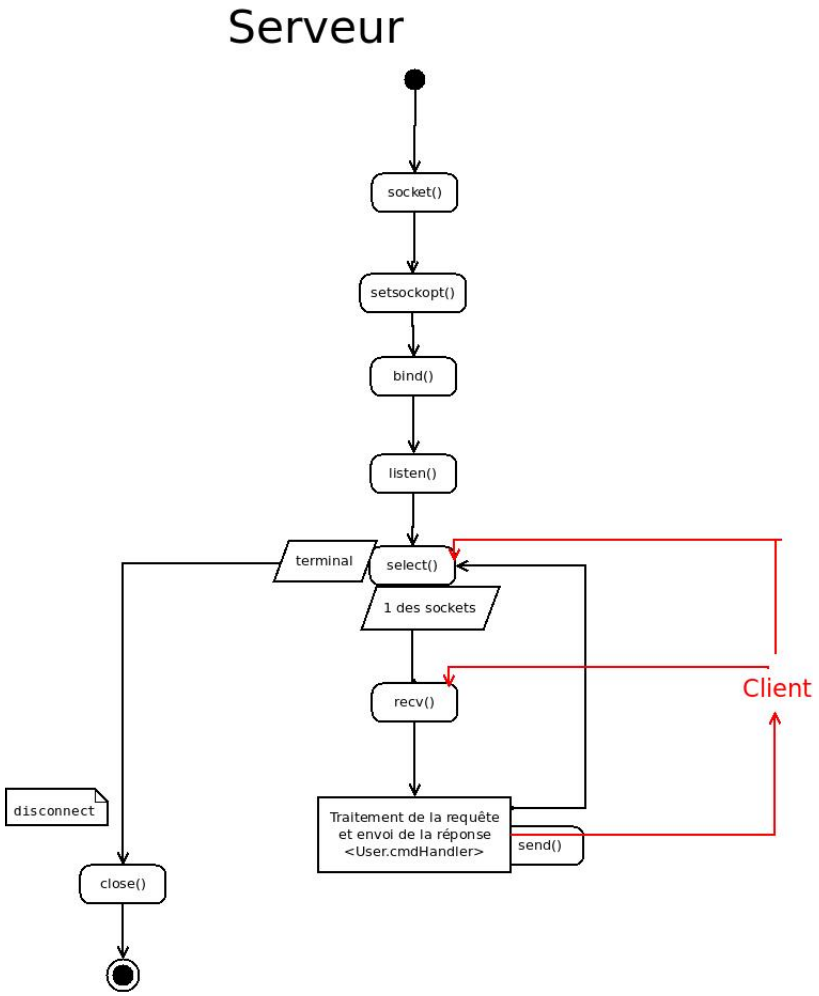

FIGURE 3.10: Diagramme d'activité réseau du serveur

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 oeuvre, 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 !). Seul le déroulement d'une vente aux enchères devra être accompagné par un thread de cadencement (un par vente).

En pratique, la séparation du code est réalisée ainsi :
– Serveur-communication : ServerMain.cpp, Server.cpp, commAPI.cpp ;

– Serveur-fonctionnel : CommonMgr.cpp, User.cpp, ...

Si les messages sont de longueur fixe prévisible, un buffer de taille fixe (INPUTSIZE) peut être utilisé de part et d'autre. Sinon (par exemple pour une liste), deux solutions sont possibles :

– Un message par ligne de la liste ;
– 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.

Pour l'instant, le code de communication est écrit pour Linux et n'est pas portable tel quel vers Windows (mais une version multi-compatible est réalisable...).
Améliorations prévues dans le futur : gestion du terminal du serveur, détection des sockets morts (clients qui n'ont pas fait disconnect suite à un crash par ex.), ...

3.3.5 Synopsis du client

Comme le serveur, le client utilise la fonction select() afin de pouvoir capter indifféremment des messages d'origine différente, en l'occurence en provenance du clavier et du socket de communication, ceci pour pouvoir recevoir des messages non sollicités.

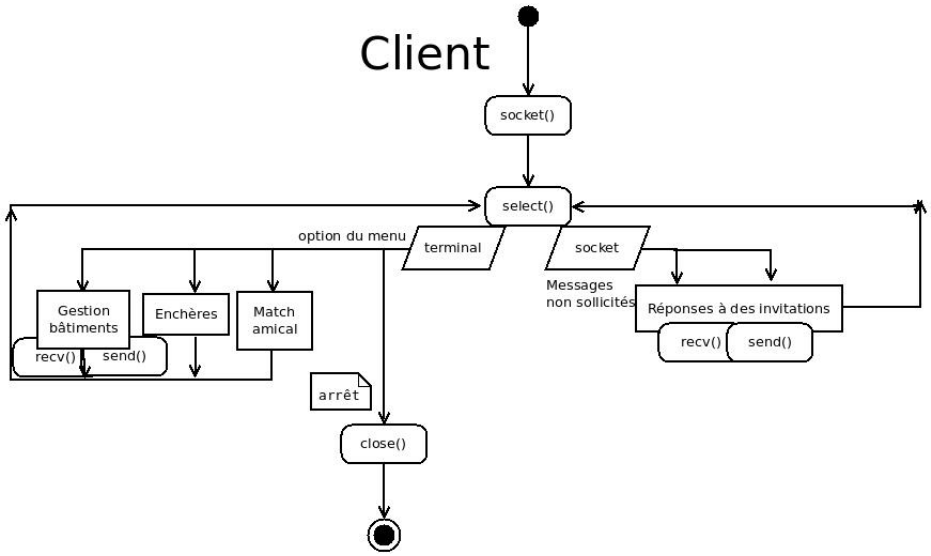

FIGURE 3.11: Diagramme d'activité réseau du client

Mais cette faculté ne peut exister que lorsque le client n'est pas occupé dans une activité particulière, c'est-à-dire lorsqu'il se voit proposer le menu (state=FREE).
Le client est donc, tout comme son image User chez le serveur, contrôlé par une variable d'état et les deux variables d'état évoluent de part et d'autre de concert.

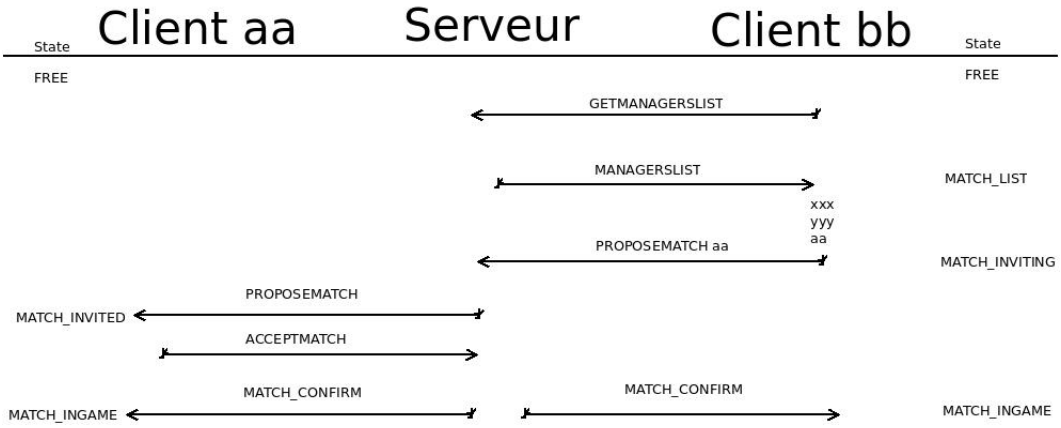

FIGURE 3.12: Exemple de diagramme d'interaction client-serveur

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 : ne peuvent par exemple être invités à un match que les managers dont state=FREE.
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 (voir par exemple la figure 3.12).
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.
On constate donc (voir figure 3.11) 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.

Cette structure sera celle que nous généraliserons lors de la mise en œuvre de l'interface graphique !

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

4 Index