

SRD : Tower Defense

Meat war : Rise of the fallen vegan

[Cours : INFO-F-209]

JACOBS Alexandre & INNOCENT Antoine & ANDRÉ Bob & VANGEEM Nicolas & ROOS Kim & PAQUET Michael &
SINGH Sundeep
BA2 Informatique

Mars 2017

Table des matières

1	Introduction	2
1.1	But du projet	2
1.2	Présentation du jeu	2
1.3	Glossaire	2
1.4	Historique de document	3
2	Besoins de l'utilisateur	4
2.1	Exigences fonctionnelles	5
2.1.1	Use case : menu	5
2.1.2	Diagramme d'activité du menu :	6
2.2	Use case : en jeu	7
2.2.1	Use case diagramme en jeu :	7
2.2.2	Diagramme d'activité lors que le joueur est en jeu :	8
2.3	Exigences non fonctionnelles	9
2.4	Exigence de domaine	9
3	Besoin du système	10
3.1	Exigences fonctionnelles	10
3.2	Exigences non fonctionnelles	10
3.3	Design et fonctionnement du système	10
3.3.1	Diagramme de sequence d'une connection	12

1 Introduction

1.1 But du projet

Notre projet consiste en une réalisation d'un qui sera jouable en réseau sous Linux. Il est aussi de permettre aux étudiants de mettre en pratique les différents concepts vus dans les cours de système d'exploitation et d'analyse et méthodologie informatiques. De plus, le respect de conventions de codage, du paradigme orienté-objet et l'analyse UML faites seront un aspect essentiel à la réalisation de notre projet.

Ce document joue le rôle du document de spécification des besoins, SRD (System requirement Document) pour le projet INFO-F209 : "Meat war : Rise of the falling vegan". Ces exigences portent sur un travail de groupe (8 personnes) codé majoritairement en C++ avec une section C pour le serveur. Le produit final est donc un jeu de type Tower Defense multijoueur en réseau. Ce jeu n'ayant aucun but lucratif, nous considérons le principal comme étant le joueur du jeu en question. Ainsi, ce document présente des spécifications propres à l'utilisateur et propre au système. Celui-ci va également mettre en avant la communication client / serveur afin de bien mettre en place l'aspect multijoueur réseau du produit.

Ce document ne varie pas en fonction des différentes architectures ou systèmes d'exploitation.

1.2 Présentation du jeu

Le nom de notre projet est "Meat war : Rise of the fallen vegan". Lors d'une création de compte, chaque utilisateur reçoit une certaine somme d'argent qui lui permettra d'acheter des tours lors des parties. Lors de partie, chaque joueur se voit attribuer une partie de la map. Chaque joueur utilisant à sa guise la partie de la map lui ayant été attribué. Par ailleurs, les termes utilisés et liés au jeu ont été choisis en fonction du thème que nous sommes imposés.

1.3 Glossaire

argent Monnaie qui permet au joueur de s'acheter des tours pour permettre de résister aux vagues d'ennemis..

client Programme que l'utilisateur lance pour se connecter au serveur. Il permet à l'utilisateur de jouer au jeu via le terminal ou l'interface graphique.

compte Contient toutes les données de l'utilisateur.

ennemi Personnage fictif qui apparaît avec intervalle régulier dans une vague qui a un certain niveau de vie..

joueur Personnage fictif du jeu qu'incarne l'utilisateur.

map Espace représentant le plateau de jeu avec lequel les joueurs pourront avoir des interactions (placer des tours sur les parties délimitées par des # et montreras aussi le déplacement des ennemis selon des chemins prédéfinies..

serveur Programme qui réceptionne les connections des clients. Il a les données de tous les utilisateurs.

supporter Personnage fictif que l'utilisateur qui ne peut pas jouer une partie mais peut intervenir en donnant de l'argent à un joueur ou plusieurs..

tours Objet de défense permettant à un joueur de contrer les vagues d'ennemis, à chaque ennemi détruit le joueur gagne de l'argent. Une tour coûte 200 ..

Tower Defense Jeu de stratégie où le but est de défendre la zone d'un joueur contre des vagues successives d'ennemis en construisant et en améliorant progressivement ses tours..

utilisateur Personnage réel qui joue à Meatwars: Revenge of the Falling Vegan.

1.4 Historique de document

Versions	Auteur	Date	Modifications
2.0	Équipe	05-03-2017	Deadline pour la phase 2
1.5	Alexandre	05-03-2017	Mise à jour diagramme UML
1.4	Alexandre	28-02-2017	Révision besoin système & utilisateur.
1.3	Alexandre	28-02-2017	Ajout de certains termes.
1.2	Alexandre	27-02-2017	Modification de la section Introduction.
1.1	Antoine & Bob	10-02-2016	Correction suite à de la réunion du 06-02-2017.
1.0	Équipe	19-12-2016	Deadline pour la phase 1
0.3	Kim	16-12-2016	Ajout du glossaire et de l'index
0.2	Équipe	16-12-2016	Ajout des besoins d'utilisateur et besoins du système.
0.1	Sundeeep & Michael	10-12-2016	Ajout des use cases et diagrammes d'activité au niveau utilisateur.
0.0	Alexandre & Antoine	10-12-2016	Création de la structure du document.

2 Besoins de l'utilisateur

Les besoins de l'utilisateur seront expliqués au travers de plusieurs graphes/explications, en voici ici un résumé. Le but de l'application est de permettre à l'utilisateur de pouvoir jouer à un jeu de type Tower Defense . Celui-ci sera disponible depuis un menu accessible après connexion et offrira au utilisateur plusieurs services. En effet, l'utilisateur devra se créer un compte avant d'avoir accès au menu.

Il pourra alors consulter son profil. Un classement sera également accessible et permettra de situer un joueur par rapport à un autre en fonction de leur score respectif.

Comme dit implicitement plus haut, l'utilisateur disposera d'une liste d'amis qui pourra modifier(ajouter, supprimer et accepter des demandes d'amis) et afficher sa liste d'amis.

Le menu permettra bien évidemment à un utilisateur premièrement de se connecter à un compte déjà existant ou de se créer un compte, deuxièmement une fois l'utilisateur connecté, ce dernier pourra décider de démarrer une partie, donc de choisir un mode de jeu pour qui ouvrira un salon en attendant d'avoir le nombre requis de joueurs afin de jouer une partie. Mode, et nombre de joueurs (au maximum de 4) seront spécifiés par le client avant que celui-ci ne lance la partie. Il pourra aussi rejoindre une partie en tant que supporter, afficher son profil et gérer ses amis. Une seule et unique carte sera cependant disponible.

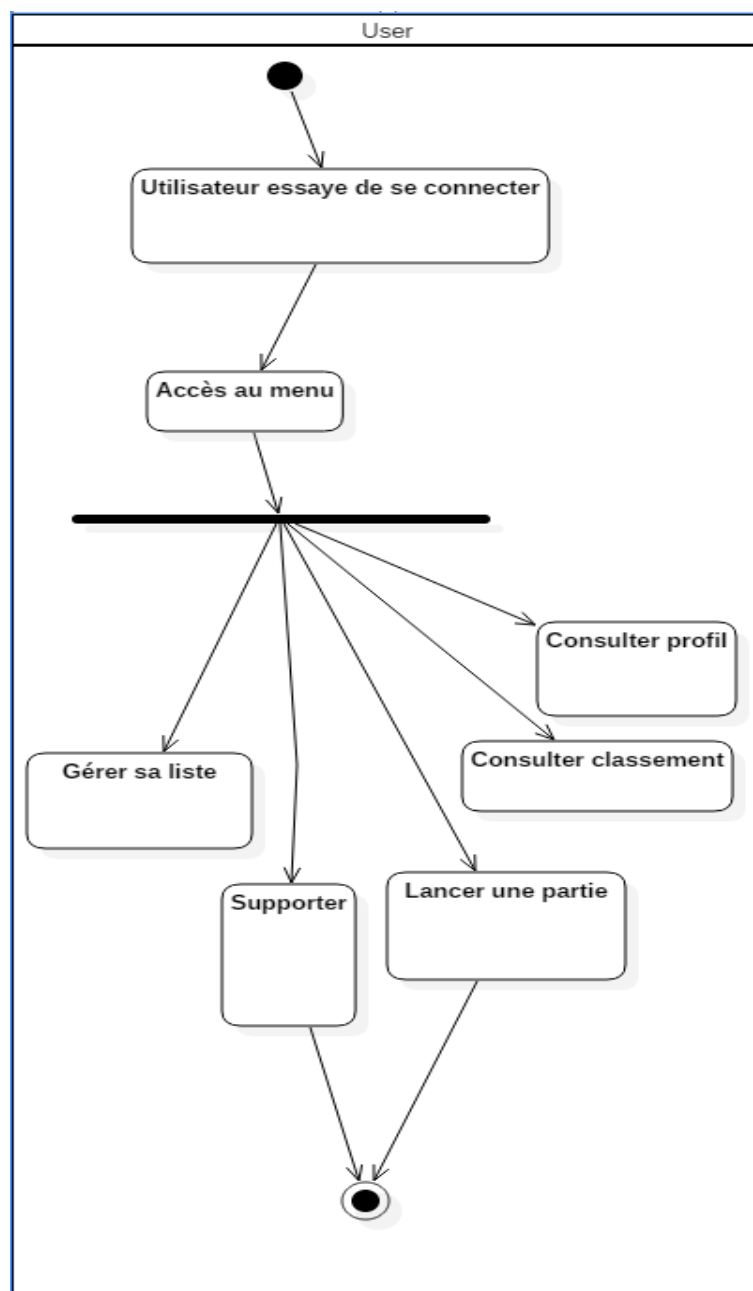
2.1 Exigences fonctionnelles

2.1.1 Use case : menu



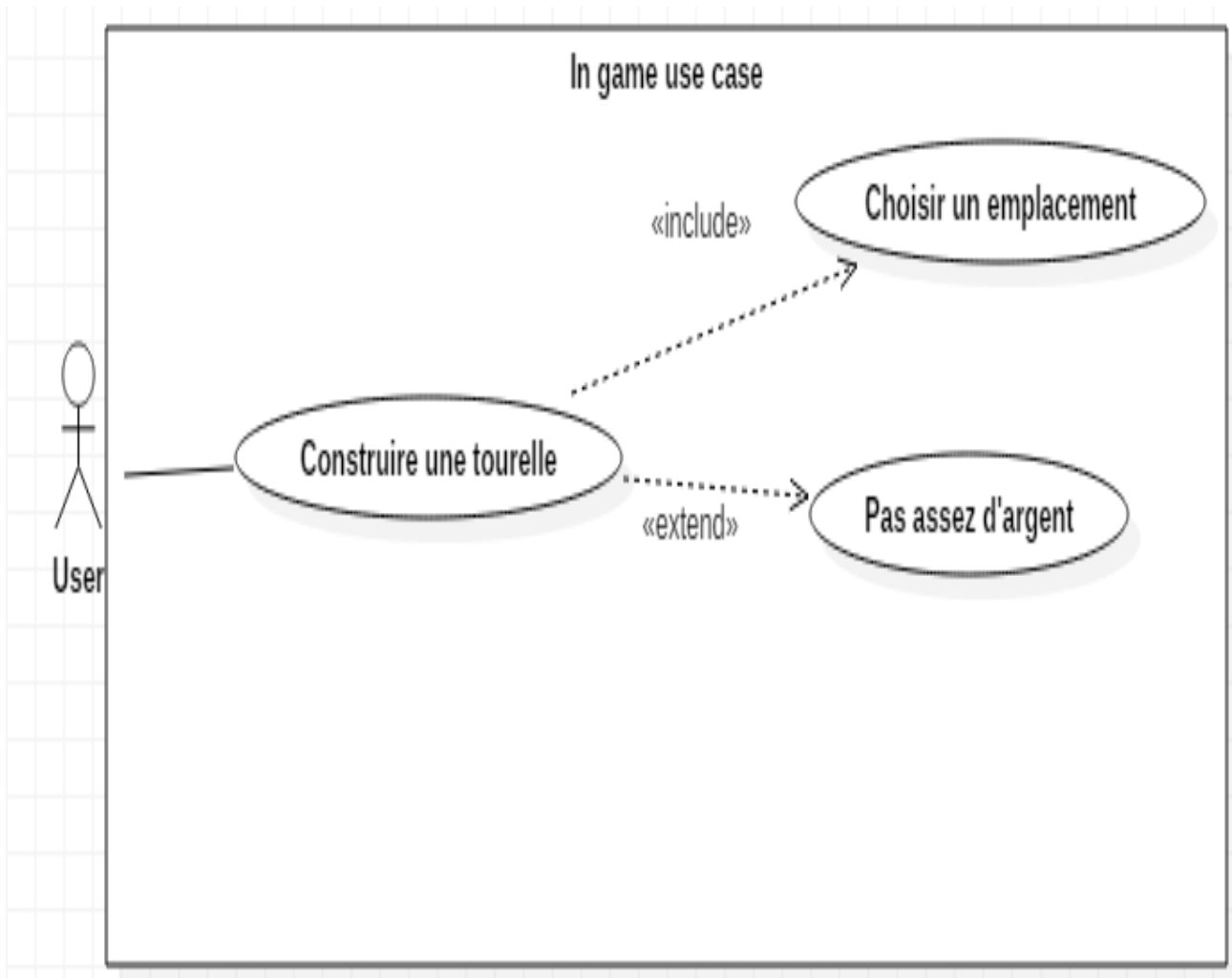
- **Acteurs** : ce use case a comme 1 acteur acteur qui est l'utilisateur.
- **Pré-conditions** : il y en a 2 , à savoir que le utilisateur se soit connecté, si c'est un nouvel utilisateur doit d'abord s'inscrire puis se connecter.
- **Post-conditions** : il y a deux post-conditions. Soit le menu est quitté et donc le joueur est déconnecté, soit le joueur a lancé une partie, rejoint une partie en tant que supporter, consulter son profil ou gérer sa liste d'amis.
- **Flux d'exécution** : en ce qui concerne le flux d'exécution basique, le joueur va d'abord se connecter. Une fois connecté, il peut lancer une partie, consulter son profil, son classement et gérer ses amis. Finalement, avant de lancer une partie, il peut choisir le mode de jeu.

2.1.2 Diagramme d'activité du menu :



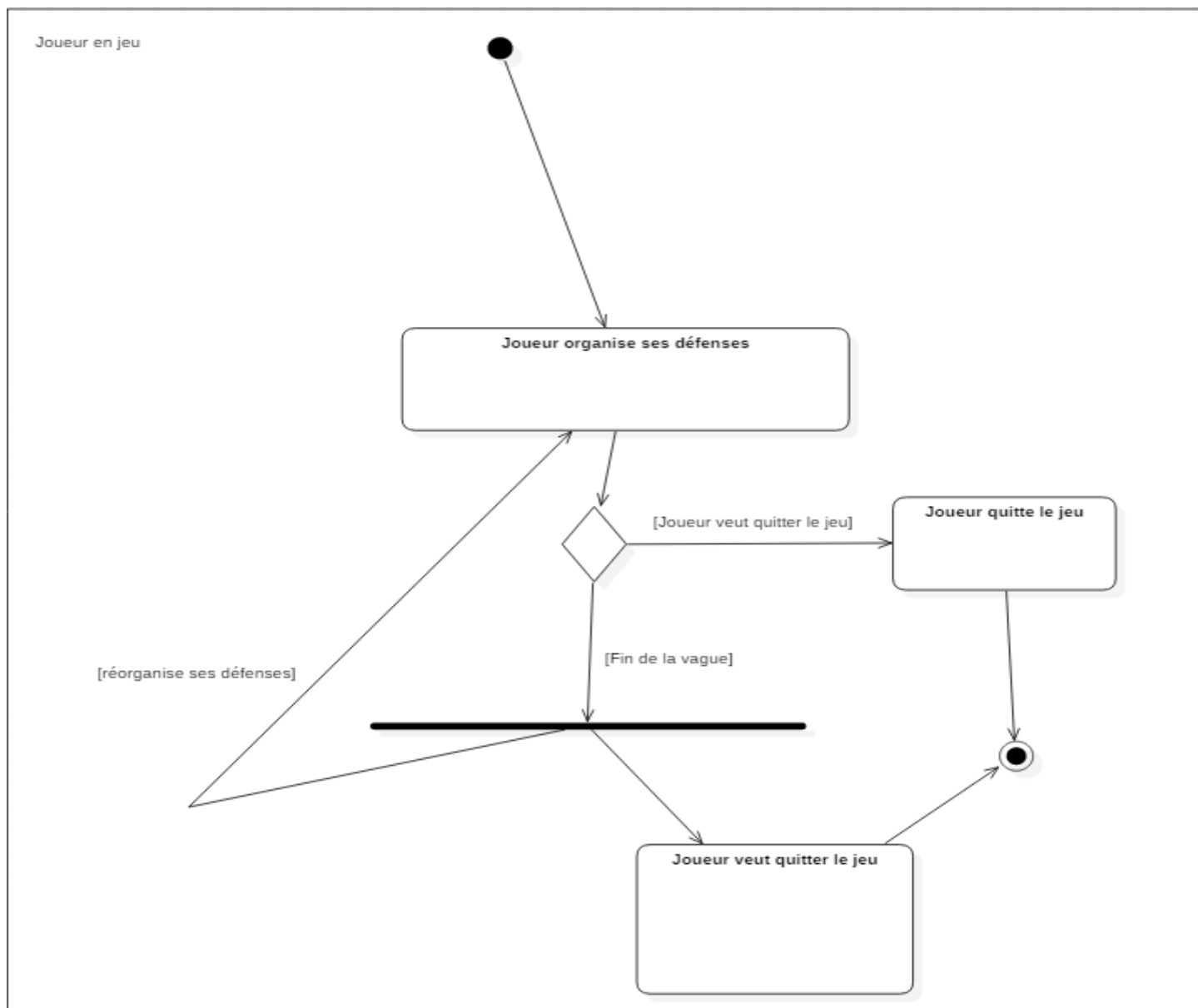
2.2 Use case : en jeu

2.2.1 Use case diagramme en jeu :



- **Acteurs** : ce use case a comme unique et principal acteur le joueur.
- **Pré-conditions** : pour ce qui est des pré-conditions le joueur doit être connecté et a dû lancer une partie.
- **Post-conditions** : la seule post-condition est qu'une fois la partie terminée, le joueur retourne au menu.
- **Flux d'exécution** : pour ce qui est du déroulement basique, une fois en jeu le joueur peut construire des tourelles, consulter la description de celles-ci, il peut également les améliorer, les vendre, mais aussi modifier les paramètres du jeu. Il existe également un flux d'exécution alternatif. Au lieu de jouer, l'utilisateur peut être tout simplement un spectateur de la partie.

2.2.2 Diagramme d'activité lors que le joueur est en jeu :



2.3 Exigences non fonctionnelles

- Menu et interface intuitifs.
- Animations graphiques.
- Maniabilité

2.4 Exigence de domaine

- Le jeu doit être multijoueur, les différents utilisateurs connectés sur un même serveur doivent pouvoir agir sur le terrain.
- L'abandon d'un des joueurs ne doit pas empêcher les autres de terminer leur partie.
- Un jeu de Tower Defense est un jeu de 2 à 4 joueurs qui ont chacun une partie de la carte.
- Chacun des joueurs a sa base dans chaque coin du jeu.
- Les ennemis apparaissent au milieu de la carte, leur vagues sont identiques pour chaque joueur.
- La partie se déroule sous forme de vagues successives d'ennemis jusqu'à ce que la partie se termine.
- Les joueurs peuvent améliorer ou acheter des tours à tout moment de la partie.
- La partie se termine si le temps est écoulé dans le cas d'une partie en mode chrono ou s'il ne reste que un joueur vivant dans le cas d'une partie classique.

3 Besoin du système

3.1 Exigences fonctionnelles

Les exigences fonctionnelles du système regroupent les besoins explicités par les clients qui ont des conséquences sur l'architecture du système. Ces exigences n'ont aucun lien avec l'utilisateur et sont donc implicites. Après avoir analysé les exigences, on a pu déterminer donc plusieurs exigences fonctionnelles :

- Démarrer&Initier une partie : Fonction qui initie une partie lorsqu'un utilisateur en fait la demande. Pour cela, différentes données sont nécessaires, telles que le mode de jeu et le nombre de joueurs requis pour la partie. Une fois le nombre atteint, la partie, le serveur lance automatiquement la partie chez tous les utilisateurs et rafraîchit leur interface.
- Trouver une partie : Lorsqu'un utilisateur veut jouer une partie ou en rejoindre une pour être supporté, le serveur fait appel au matchmaking qui permet de regrouper plusieurs personnes dans un même salon de jeu, en attendant le début d'une partie.
- Partie : Cela est une composante importante du programme. Elle se compose de diverses fonctionnalités : placer une tour, vérification de victoire, défaite et établissement des scores et récompenses attribuées à chaque joueur.
- Classement : Regroupe tous les scores des joueurs classés du meilleur au moins bon score et est mis à jour après chaque partie.

3.2 Exigences non fonctionnelles

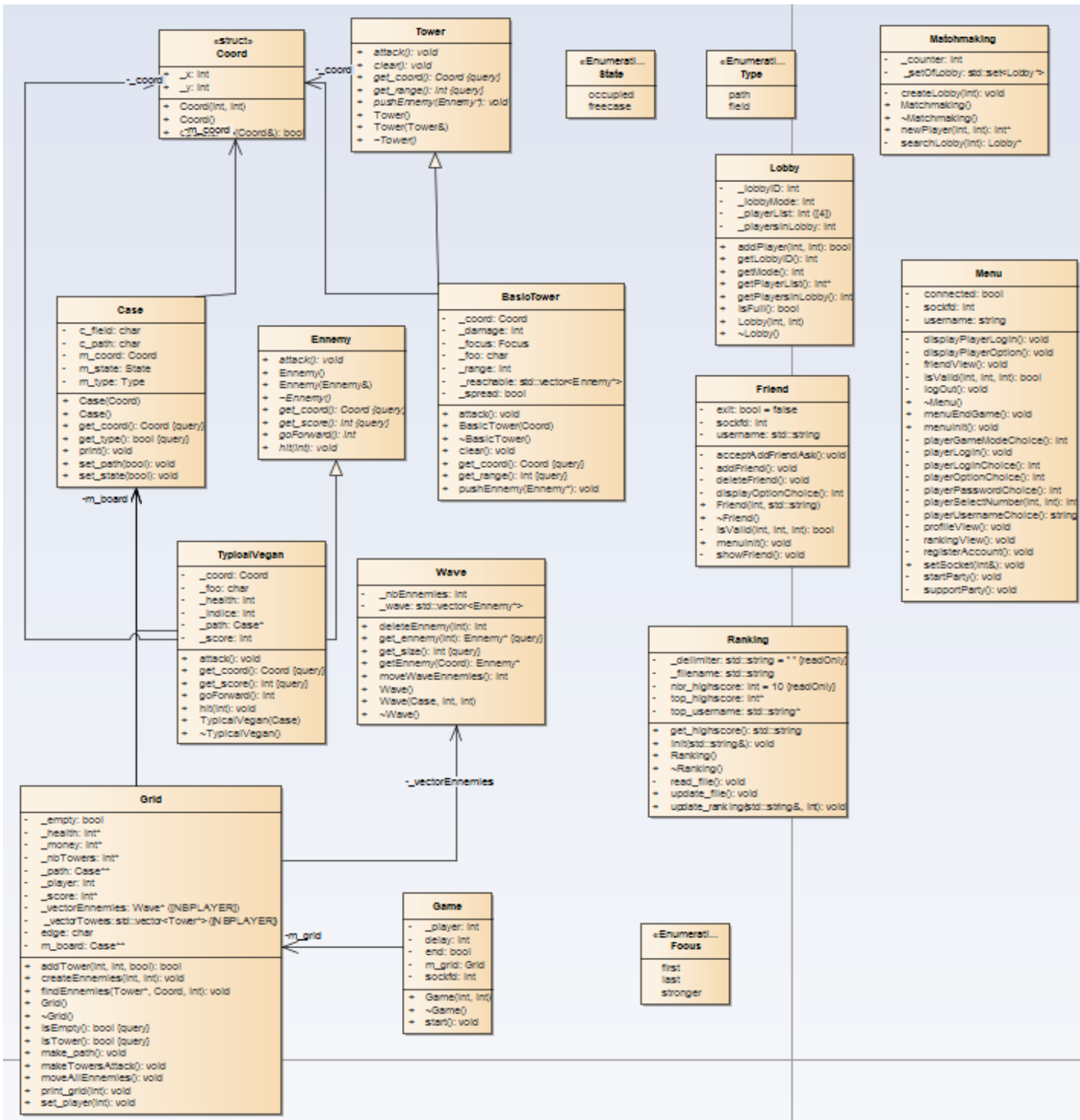
Les exigences non-fonctionnelles reprennent l'ensemble des exigences qui sont considérées comme importantes afin de réaliser correctement le projet. Dans cette section, nous reprenons donc les quelques aspects qui n'ont pas été exprimés de manière explicite par le client. Par ailleurs, il est évident que certains nombres d'interactions entre le client et le serveur seront mises en place, vu que le projet est un programme devant être jouable en réseau.

- Création d'un compte utilisateur et connexion de celui-ci au système.
- Mise à jour des amis d'un joueur.
- Organisation.
- Réactivité.

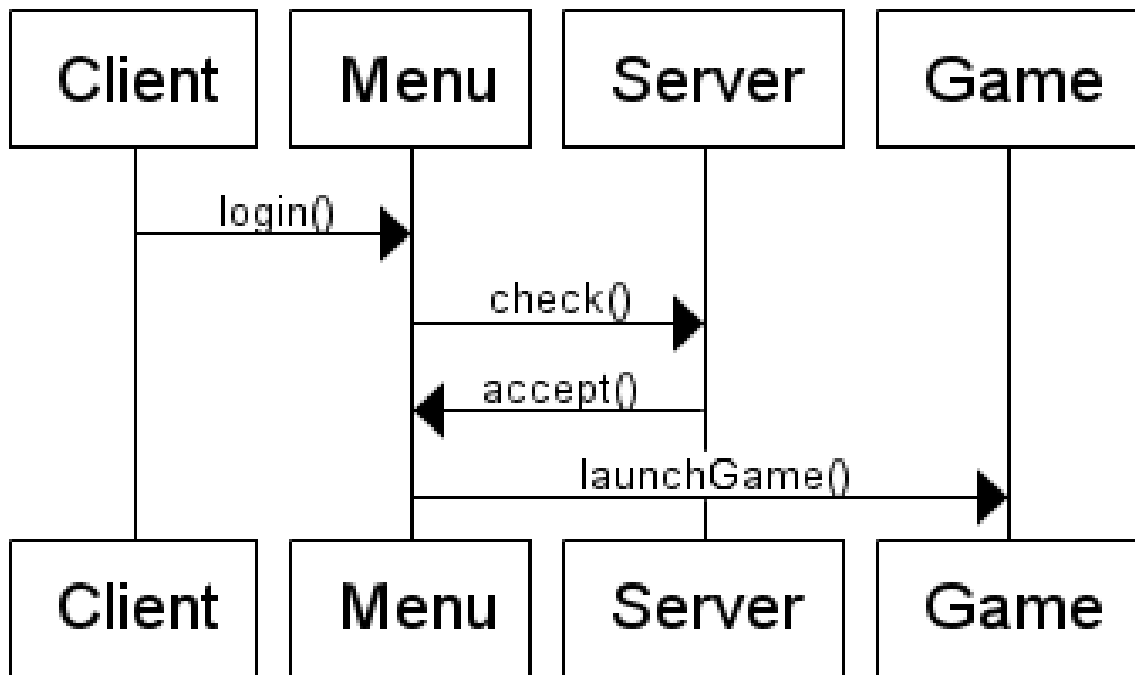
De plus, les différents points, qui sont repris ci-dessus, ne reflètent les aspects clés qui ont été déterminés suite à la rencontre avec le client.

3.3 Design et fonctionnement du système

Cette section reprend les diagrammes permettant de décrire l'architecture du système et son fonctionnement. Les différents diagrammes UML ayant été utilisés sont les diagrammes de classe, de séquence et d'activité.



3.3.1 Diagramme de sequence d'une connection



Index

Tower Defense, 2
client, 2

abandon, 9
argent, 2

base, 9

classement, 4
client, 2, 10
compte, 2, 4, 10
consulter son profil, 4

ennemi, 2, 9

joueur, 2, 4, 5, 7, 9, 10

liste d'amis, 4

map, 2
menu, 4, 5, 7, 9
mode chrono, 9

partie, 4, 5, 7, 9, 10
partie classique, 9

serveur, 2, 9, 10
supporter, 4

tours, 2
Tower Defense, 2, 4, 9

utilisateur, 2, 4, 5, 7, 9, 10