

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	3
1.1	But du projet	3
1.2	Présentation du jeu	3
1.3	Glossaire	3
1.4	Historique de document	5
2	Besoins de l'utilisateur	6
2.1	Exigences fonctionnelles	6
2.2	Exigences non fonctionnelles	11
2.3	Exigence de domaine	11
3	Besoin du système	12
3.1	Exigences fonctionnelles	12
3.2	Exigences non fonctionnelles	12
3.3	Design et fonctionnement du système	13
4	Modules	20
5	Module Match	20
5.1	Gameplay	20
5.2	Choix d'implémentation	20
5.3	Support	21
6	Communication	21
6.1	Architecture générale	21
6.1.1	Mode de fonctionnement	21
6.1.2	Architecture du client	21
6.1.3	Architecture du serveur	21
6.2	Communication inter-thread	21
7	Module stockage des données	23
7.1	Stockage des comptes	23
7.2	Stockage de la liste d'amis	23
7.3	stockage de la vague d'ennemis	23
7.4	Accès au données	24
8	Matchmaking	25

9	Affichage console	26
9.1	Inscription / connexion	26
9.2	Menu	27
9.3	Affichage du jeu	28
10	Interface graphique	29

1 Introduction

1.1 But du projet

Notre projet consiste en une réalisation d'un jeu 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 partie délimité par des # et montreras aussi le déplacement des ennemis selon des chemins prédéfinies..

pseudo Un pseudo est un nom fictif crée par un joueur.

.

qt Librairie c++ utilisée dans ce projet pour la réalisation de la partie graphique.

.

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 peut jouer une partie mais peut intervenir en donnant de l'argent à un joueur ou plusieurs..

thread Un processus possédant plusieurs tâches qui s'exécutent de manière quasi-simultané. .

tour Objet de défense permettant à un joueur de contrer les vagues d'ennemis, à chaque ennemi détruit le joueur gagne de la argent..

tourelle Une tourelle dans le cadre de ce jeu, est un item défensive placé par le joueur pour se défendre contre les vagues d'ennemis.

.

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
3.0	Équipe	30-03-2017	Deadline pour la phase 3.
2.8	Sundeeep	27-03-2017	Ajout diagramme d'activité menu.
2.7	Sundeeep	27-03-2017	Ajout diagramme d'activité inscription.
2.6	Sundeeep	27-03-2017	Ajout diagramme d'activité se connecter.
2.5	Sundeeep	27-03-2017	Ajout diagramme séquence du matchmaking.
2.4	Sundeeep	27-03-2017	Ajout Seq Diag de l'accès à une donnée.
2.3	Sundeeep	27-03-2017	Ajout descriptions des tous les modules.
2.2	Sundeeep	27-03-2017	Ajout diagramme de séquence en jeu.
2.1	Sundeeep	27-03-2017	Révision des uses case et descriptions
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 utilisateur et système
0.1	Sundeeep & Michael	10-12-2016	Ajout des use cases et diagrammes d'activité au niveau uti
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

Les besoins fonctionnels sont représentés par deux diagrammes uml.Ceux-ci sont repris ci-dessous et représentent une partie de l'application.Bien évidemment, ceux-ci sont accompagnés des post-conditions, pré-conditions...

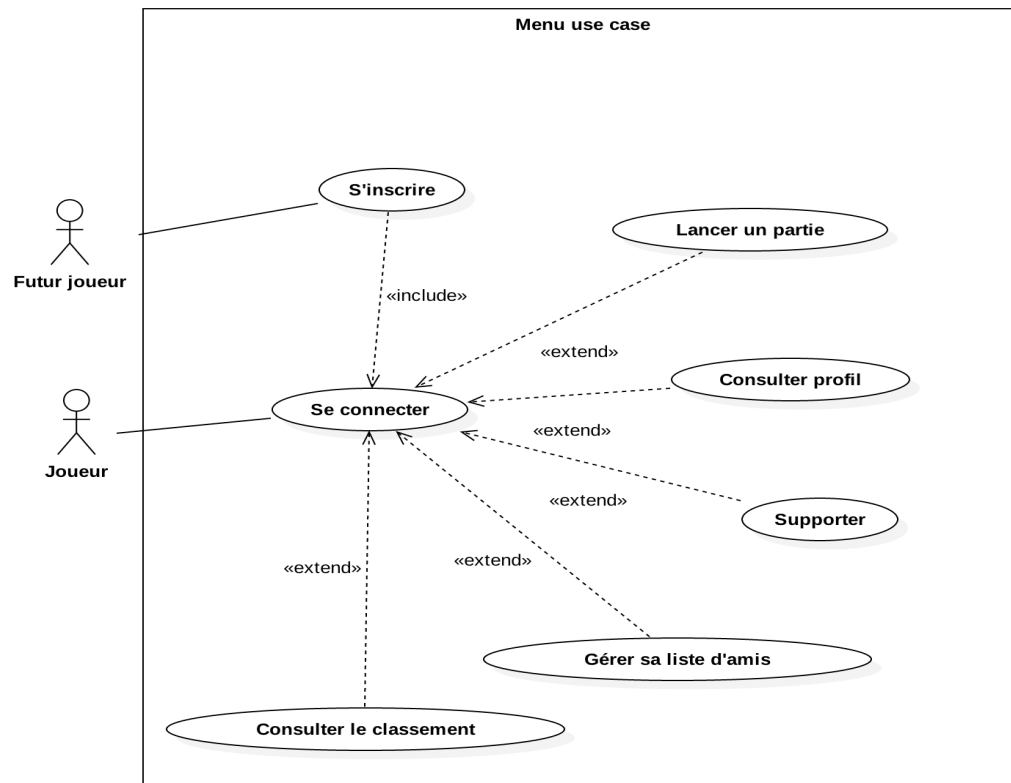


FIGURE 1 – Diagramme uml avant partie

Use case	Pré-condition	Post-condition	Flux d'exécution basique	Flux d'exécution alternatif
S'inscrire.	L'utilisateur est connecté au serveur .	L'utilisateur à un compte.	L'utilisateur crée son compte et se connecte.	Le pseudo est déjà pris.
Se connecter	L'utilisateur a un compte valide.	L'utilisateur est connecté.	L'utilisateur saisit son mot de passe et pseudo et se connecte.	Le compte est déjà connecté, Le mot de passe/pseudo est incorrect.Le jeu à atteint le nombre de joueur max.
Lancer une partie	L'utilisateur est connecté.	L'utilisateur est dans le lobby.	L'utilisateur lance une partie et rejoint un lobby.	X
Supporter	L'utilisateur est connecté.	L'utilisateur soutient un autre.	L'utilisateur sélectionne le mode support et soutient un utilisateur.	L'utilisateur ne peut pas soutenir un autre parce qu'il n'y a aucune partie lancée.
Consulter profil	L'utilisateur est connecté.	L'utilisateur consulte son profil.	L'utilisateur décide de voir son profil et celui-ci s'affiche.	X
Gérer sa liste d'amis	L'utilisateur est connecté.	L'utilisateur voit sa liste d'amis	L'utilisateur décide de voir sa liste d'amis et il va ensuite gérer celle-ci.	X
Consulter le classement	L'utilisateur est connecté.	Le joueur consulte le classement	Le joueur décide de consulter le classement et celui-ci s'affiche.	X

FIGURE 2 – Description diagramme use case avant une partie

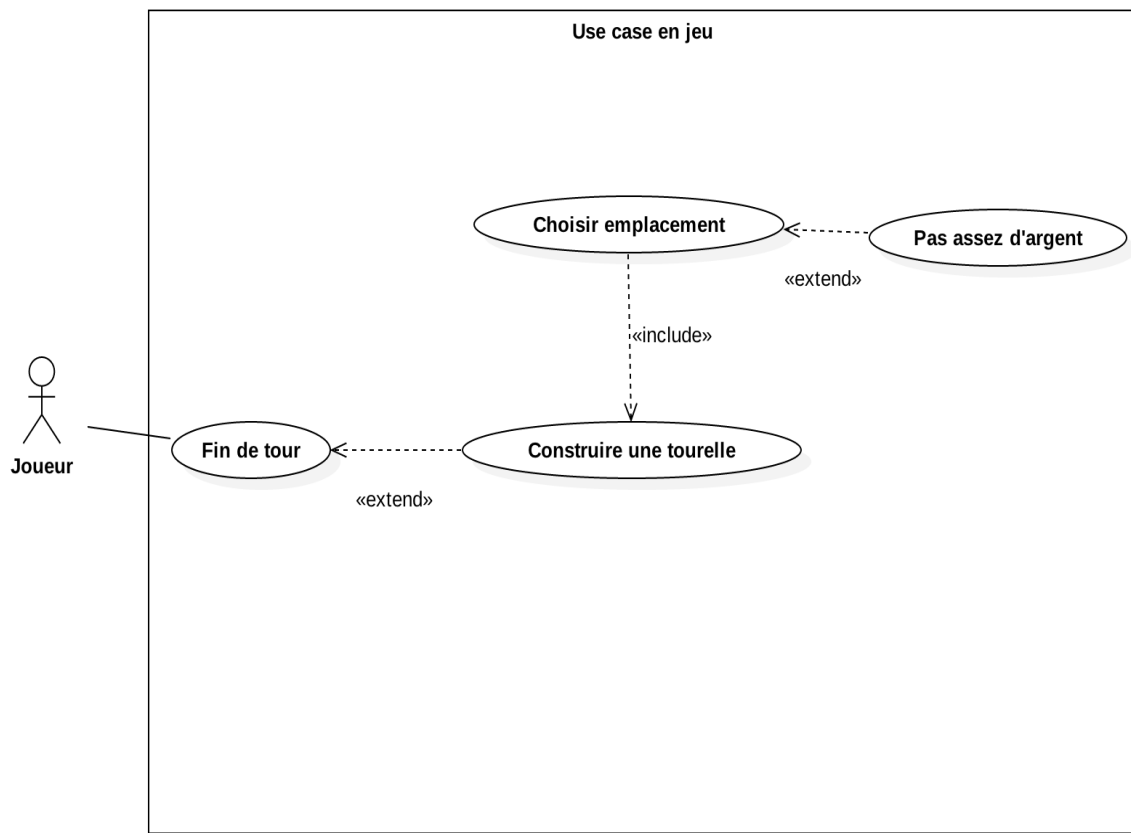


FIGURE 3 – Diagramme uml en partie

Use case	Précondition	Postcondition	Flux d'exécution basique	Flux d'exécution alternatif
Construire une tourelle	L'utilisateur est en jeu et a suffisamment d'argent.	L'utilisateur peut choisir l'emplacement de sa tourelle.	Le joueur sélectionne un type de tourelle et son argent diminue en fonction de son prix..	L'utilisateur choisit un type de tourelle et n'a pas assez d'argent.
Choisir un emplacement	L'utilisateur a su acheter une tourelle.	La tourelle est placée correctement.	L'utilisateur choisit un emplacement et place sa tourelle.	L'utilisateur choisit un emplacement mais une tourelle est déjà placée.
Pas assez d'argent	Avoir dépensé tout son argent.	X	L'utilisateur veut acheter une tourelle mais il n'a pas assez d'argent.	X
Fin de tour	L'utilisateur a survécu à la vague précédente	L'utilisateur est en plein dans une nouvelle vague.	Le joueur termine une vague et peut ensuite organiser ses défenses, acheter des tourelles ...	Le joueur se rend et c'est la fin de la partie pour lui.

FIGURE 4 – Diagramme uml en partie

2.2 Exigences non fonctionnelles

Dans le cadre de ce projet plusieurs exigences non fonctionnelles ont été relevés. Premièrement plusieurs deadlines ont été mises en place. La première le 19-12-2016, la seconde pour le dimanche 5 mars et la dernière pour le 31 mars 2017. En ce qui concerne les langages de programmation imposés, il s'agit du C et du langage C++. De même, le projet doit être capable de fonctionner sur plusieurs plateformes telles que linux, window et mac os. Les autres besoins relevés concernent plus le jeu et sont ceux-ci :

- ***Menu et interface intuitifs*** :

En effet, un menu ou une interface peu intuitif nuit très vite à l'expérience de l'utilisateur. Il est donc impérative que celui-ci se retrouve très facilement.

- ***Animations graphiques*** :

De même, un jeu parfaitement bien animé, plaisant à regarder améliore l'expérience apportée par celui-ci.

- ***Maniabilité*** :

Outre le fait que l'interface doit être intuitive, il est également important d'avoir un jeu ayant une bonne maniabilité.

2.3 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, les 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 qu'un joueur vivant dans le cas d'une partie classique.

3 Besoin du système

3.1 Exigences fonctionnelles

Les exigences fonctionnelles du système sont les besoins expliqués par le client mais qui n'ont aucun lien avec celui-ci. Elles sont donc implicites et après analyses des exigences nous avons pu discerner 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 tels que le mode de jeu et le nombre de joueurs requis pour la partie. Une fois le nombre de joueur atteint pour une 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 le soutien d'un autre utilisateur, le serveur fait appel au matchmaking qui permet de regrouper plusieurs personnes dans un même mode de jeu, en attendant le début d'une partie. La partie commençant quand elle est pleine.
- **Partie** :
Cela est une composante importante du programme. La partie se compose de diverses fonctionnalités : placer une tour, vérification de victoire, défaite et établissement des scores et récompenses attribués 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 sont très importantes pour réaliser correctement le projet. Cette section reprend donc un certain nombre d'aspects qui n'ont pas été déclarés clairement 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.

Voici quelques-unes de ces interactions :

- 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 que les aspects clés qui ont été déterminés suite à la rencontre avec le client.

3.3 Design et fonctionnement du système

Cette sections reprends les diagrammes permettant de décrire l'architecture du système et son fonctionnement. Les différents diagrammes UML ayant été utilisé sont diagrammes de classe, de séquence et d'activité. Veuillez consulter la section uml pour voir ses diagrammes en entier.

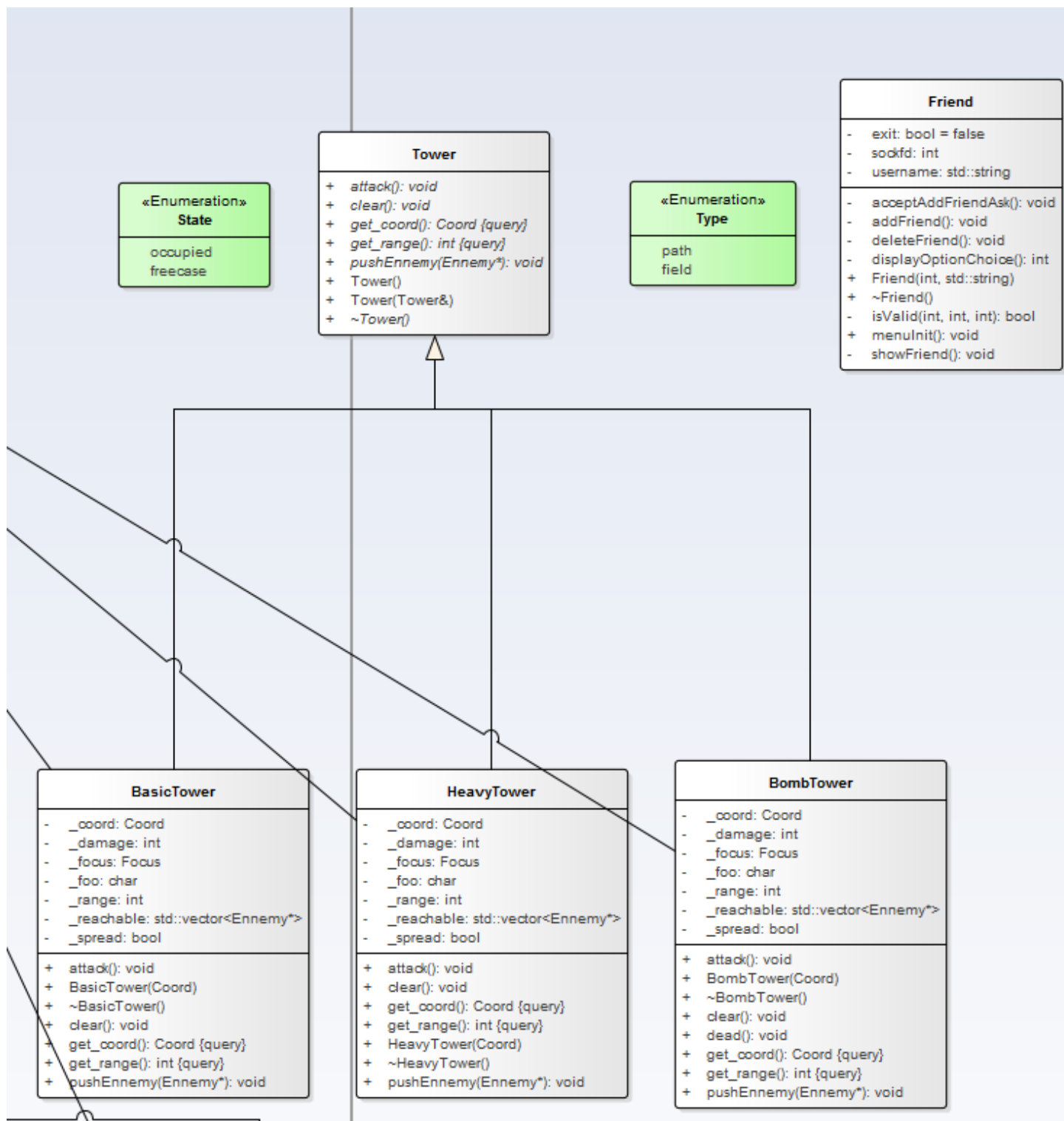
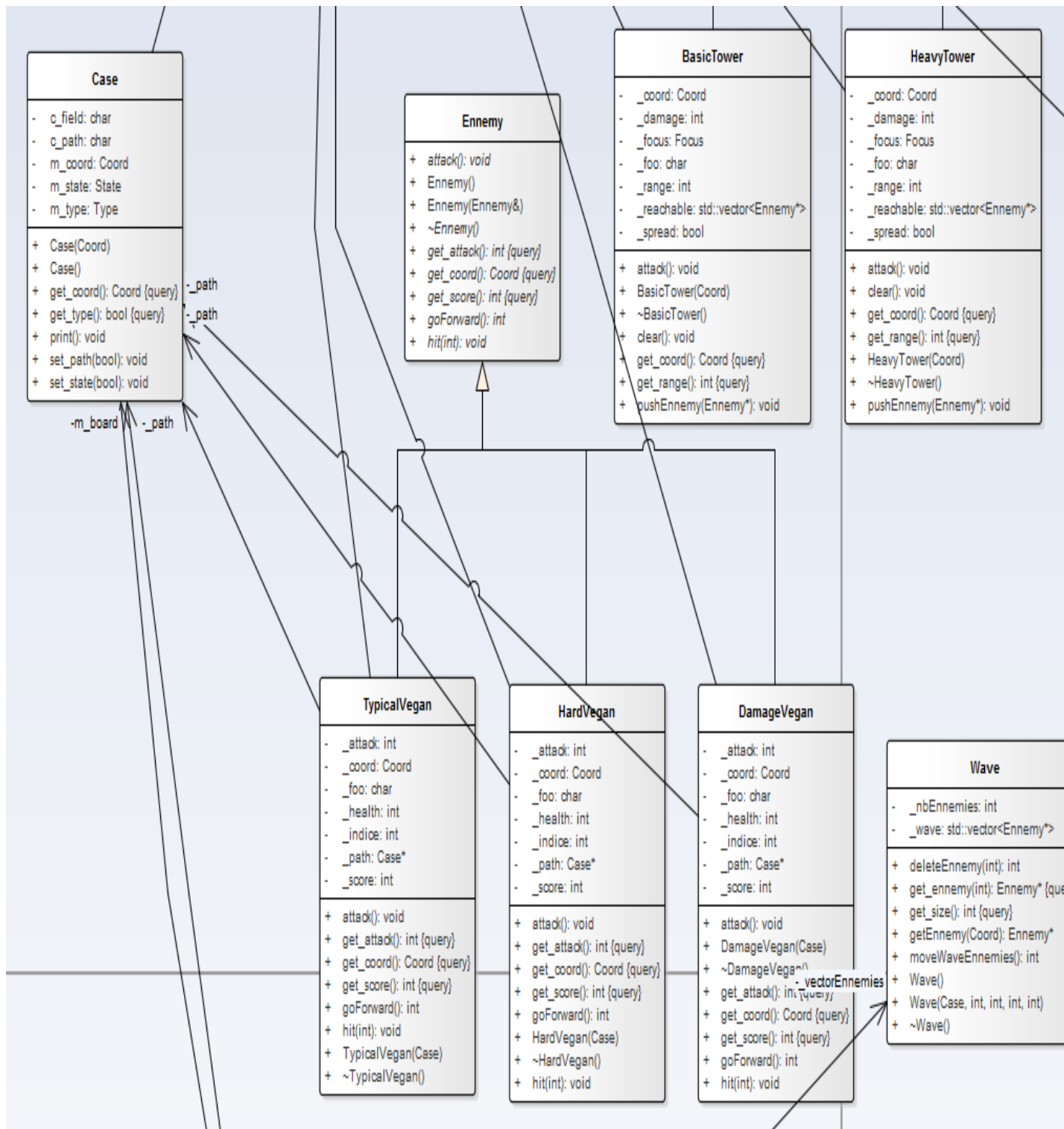
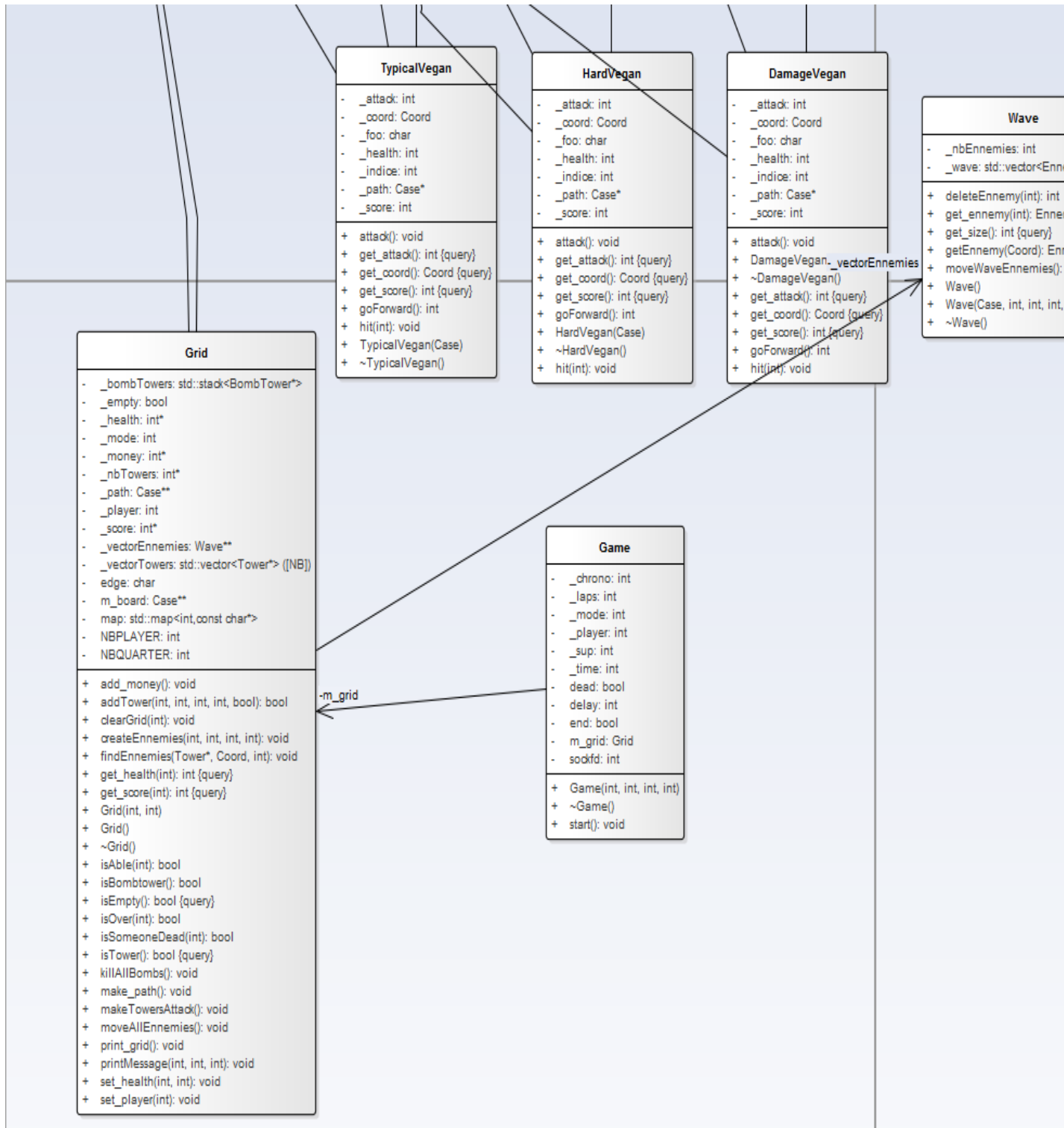
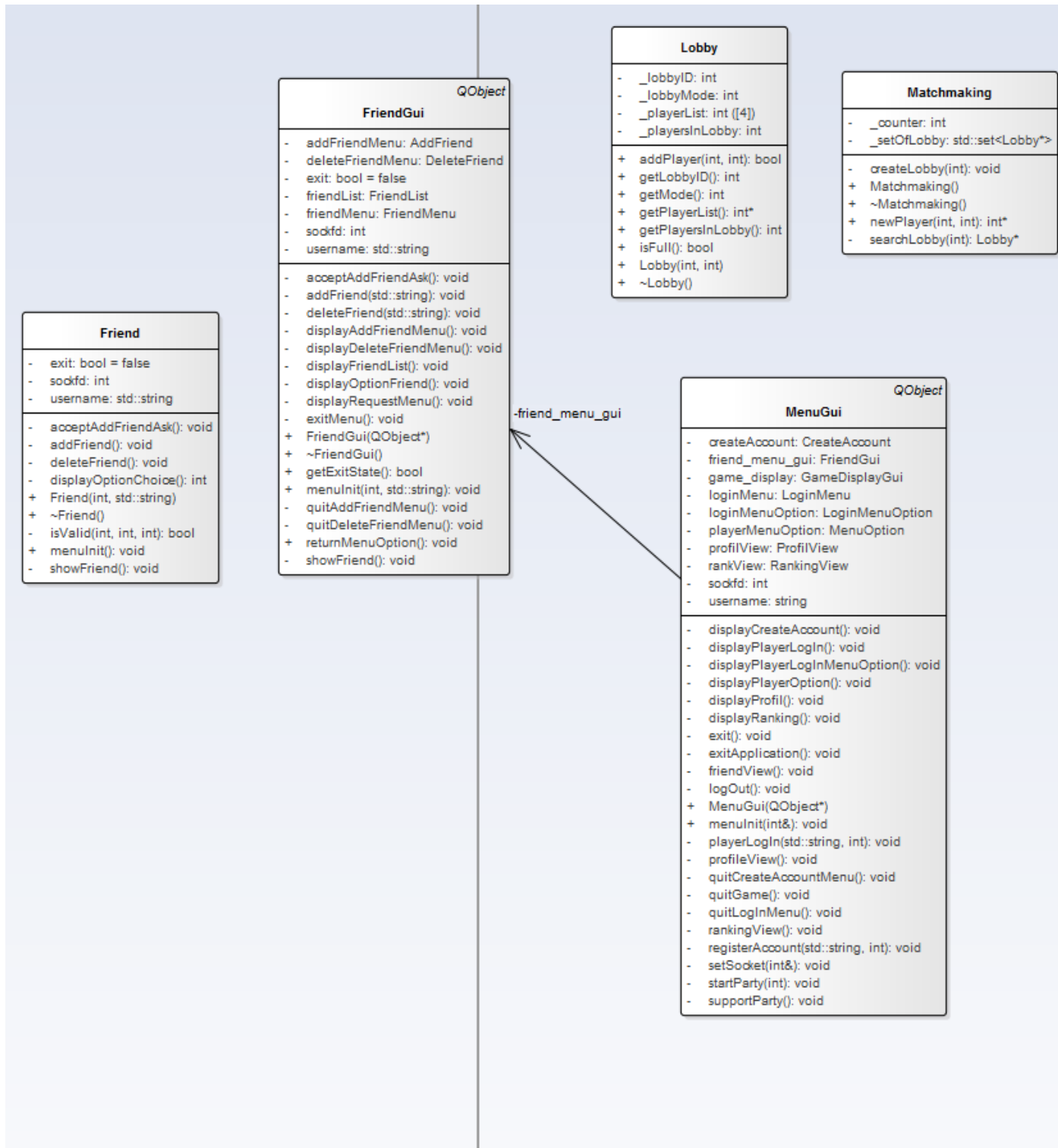


FIGURE 5 – Diagramme de classe partie terminale









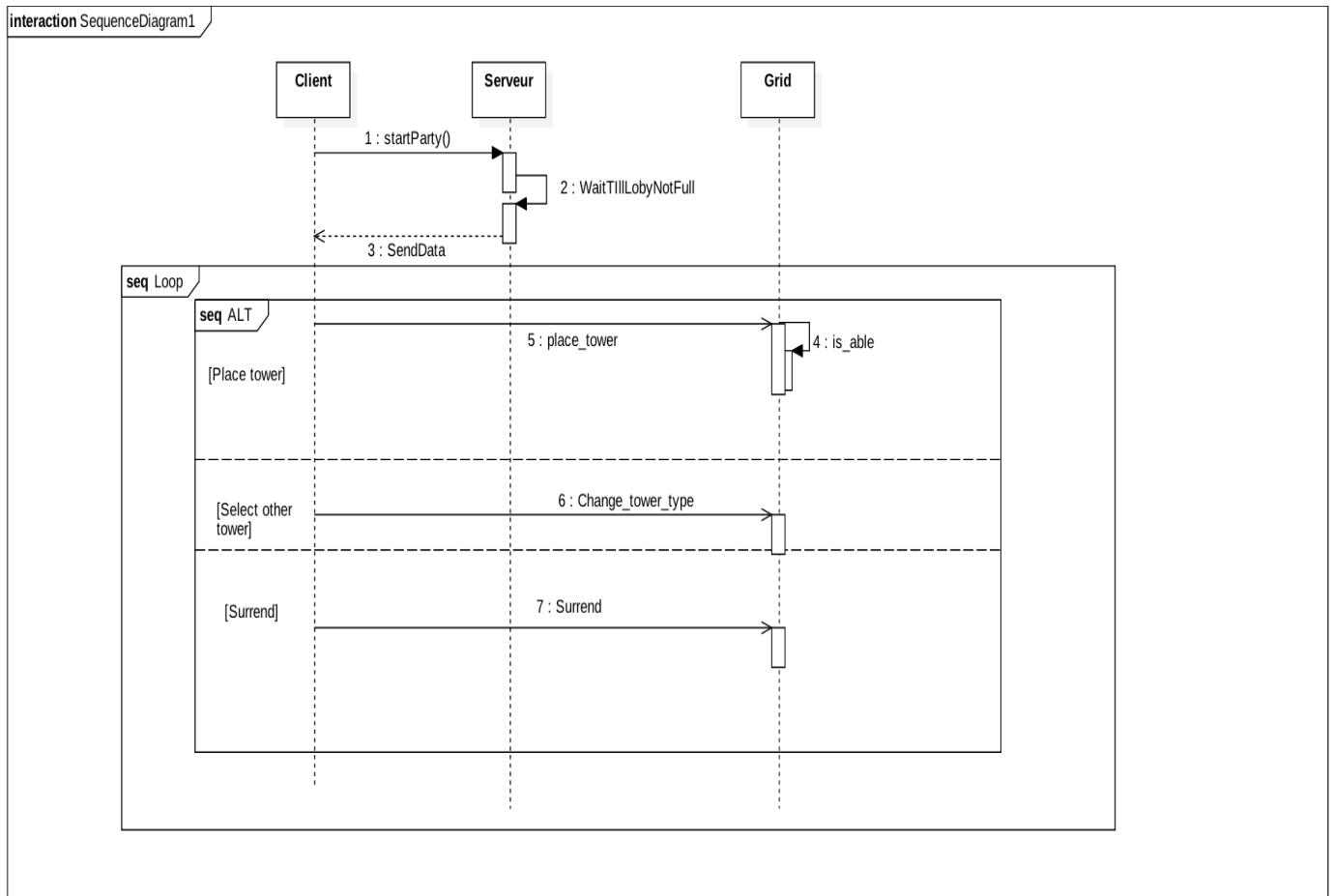


FIGURE 6 – Diagramme de sequence en jeu

4 Modules

Pour faciliter le développement du projet mais aussi pour avoir une certaine cohérence, le programme a été découpé en plusieurs modules. Ceux-ci sont détaillés ci-dessous.

5 Module Match

Cette section explique la façon dont le jeu est joué et la manière dont on a implémenté le gameplay.

5.1 Gameplay

Entre chaque vague l'utilisateur à la possibilité de :

- Placer une tourelle.
- Surrend (abandonner la partie).

Parmi les tourelles disponibles l'utilisateur peut placer 3 types de tourelles :

- Basique coûtant 100 "Money".
- Heavy tourelle plus chère mais plus puissante et coûtant 200 "Money".
- Bomb tourelle qui dure qu'une seule vague mais dévastatrice avec un prix de 300 "Money".

Bien évidemment le joueur peut placer autant de tourelle qu'il veut, tant que celui-ci a assez de "Money" et dans les limites du temps entre chaque vagues.

Le terrain est séparé en 4 parties pour 4 joueurs et chacune de ces parties comportent un couloir par lequel des vagues de monstres apparaissent. Les monstres sont représentés par différents symboles et clignotent en rouge lorsqu'ils se font toucher par une tourelle.

5.2 Choix d'implémentation

Cette partie décrit des choix personnels quant au gameplay et donc non explicités par le client. Ceux-ci sont repris ci-dessous.

- Tourelle : celles-ci regardent à chaque moment si des ennemis se trouvent à leur porté et si le choix de la cible se porte vers l'ennemi le plus proche de la base.
- Ennemi : lorsqu'un ennemi est en mouvement celui-ci vérifie sa vie. Si elle est à zéro suite à l'attaque d'une tourelle, ils disparaissent, sinon ils continuent d'avancer jusque la base.

5.3 Support

Outre le fait qu'un support est un utilisateur comme un autre, celui-ci ne joue pas vraiment une partie et rejoint quand une partie est déjà lancée. De manière générale, le support indique le pseudo de l'utilisateur qu'il souhaite soutenir et est placé dans une queue. Suite à ça, si l'utilisateur que le support souhaite soutenir est en jeu, L'utilisateur étant supporté gagnera automatiquement 100 de "Money".Finalement, le support devra attendre la fin de la partie et ne pourra donc plus rien faire.

6 Communication

6.1 Architecture générale

6.1.1 Mode de fonctionnement

L'idée générale pour gérer cette communication entre le client et le serveur est la mise en place de nombreux `recv` et `send`. De cette manière la communication entre les deux suit une certaine logique. Cette façon de communiquer peut paraître dangereuse à cause d'un `recv` bloquant mais il n'en est rien. Un soin particulier à été apporté à l'emplacement des envoies et des `recv`, de sorte à ce que la communication entre le client et le serveur soit la plus fluide possible.

6.1.2 Architecture du client

De la même manière, le client fait appel à différents objets qui vont directement communiquer avec le serveur via des `send` et des `recv`. Le tout étant encore une fois agencé avec précaution et une certaine logique, de manière à ce que la communication client-serveur se fasse le plus facilement possible.

6.1.3 Architecture du serveur

Le serveur va dédié un thread à chaque fois qu'un nouveau utilisateur va faire une nouvelle demande de connexion.De cette manière, le serveur garantis la même interaction quel que soit l' utilisateur. De même, le serveur est celui-ci qui se charge de toute les requêtes envoyés par le client.Celui-ci se charge donc de l'écriture et lecture dans un fichier.

6.2 Communication inter-thread

Une communication entre thread est nécessaire lorsque tous les utilisateurs sont en jeu. En fonction de l'action de l'utilisateur le serveur va recevoir plusieurs informations.

- Placement d'une tourelle : coordonnées en x et y, l'id de l'utilisateur. et le type de la tourelle.
- Surrender : des coordonnées et l'id de l'utilisateur .

Une fois les informations obtenues, le serveur va automatiquement envoyer les données à tous les autres utilisateurs. Par conséquent, tous les utilisateurs voient ce que les autres utilisateurs font à tout moment de la partie.

7 Module stockage des données

7.1 Stockage des comptes

Pour sauvegarder toutes les données telles que :

- le pseudo
- le mot de passe
- une valeur indiquant l'état de l'utilisateur(connecté ou non)

nous avons donc créé un fichier texte appelé "account.txt" qui gardera donc à tout moment ces données. Le but premier du projet n'étant pas la sécurité mais bien un jeu fonctionnel, nous avons décidé de mettre en place un système de fichier texte et non une base de données car nous sommes plus familiarisé avec un système de fichiers. Bien évidemment si le fichier n'existe pas, il sera créé par le serveur.

7.2 Stockage de la liste d'amis

De même, pour chaque utilisateur un fichier texte sous son nom est crée pour contenir sa liste d'amis. Une demande d'ami est représenté dans le fichier par une valeur "0" à côté du pseudo de l'utilisateur ayant envoyé la demande. Ce fichier sous la forme "NomUtilisateur.txt" va servir pour de nombreuses opérations telle que :

- La visualisation du profil.
- La consultation de la liste d'amis.
- accepter un amis.
- ajouter un amis.
- supprimer un amis.

Une valeur "1" signifie que les deux utilisateurs sont déjà amis.

7.3 stockage de la vague d'ennemis

Tout comme pour les comptes et la liste d'amis, nous avons décider de continuer sur un système de fichier, et nous avons donc créer un fichier pour stocker les vagues d'ennemis. Plus précisément, celui-ci est composé de 4 champs qui indiquent :

1. Le nombre total d'ennemis par vague.
2. le nombre total d'ennemis de type "a".
3. le nombre total d'ennemis de type "b".
4. le nombre total d'ennemis de type "c".

7.4 Accès au données

Finalement, le client ne contient donc aucune données. À chaque requêtes telles qu'une demande d'ami, consulter son profil... Celui-ci doit passer par le serveur pour recevoir une réponse. Cette façon de faire bien que coûteuse en performance(lecture dans les fichiers, transfert d'informations entre le client et le serveur ...) garantit une séparation entre le tâche du client et du serveur . Si l'utilisateur demande de se connecter, la vérification passera donc par le serveur.

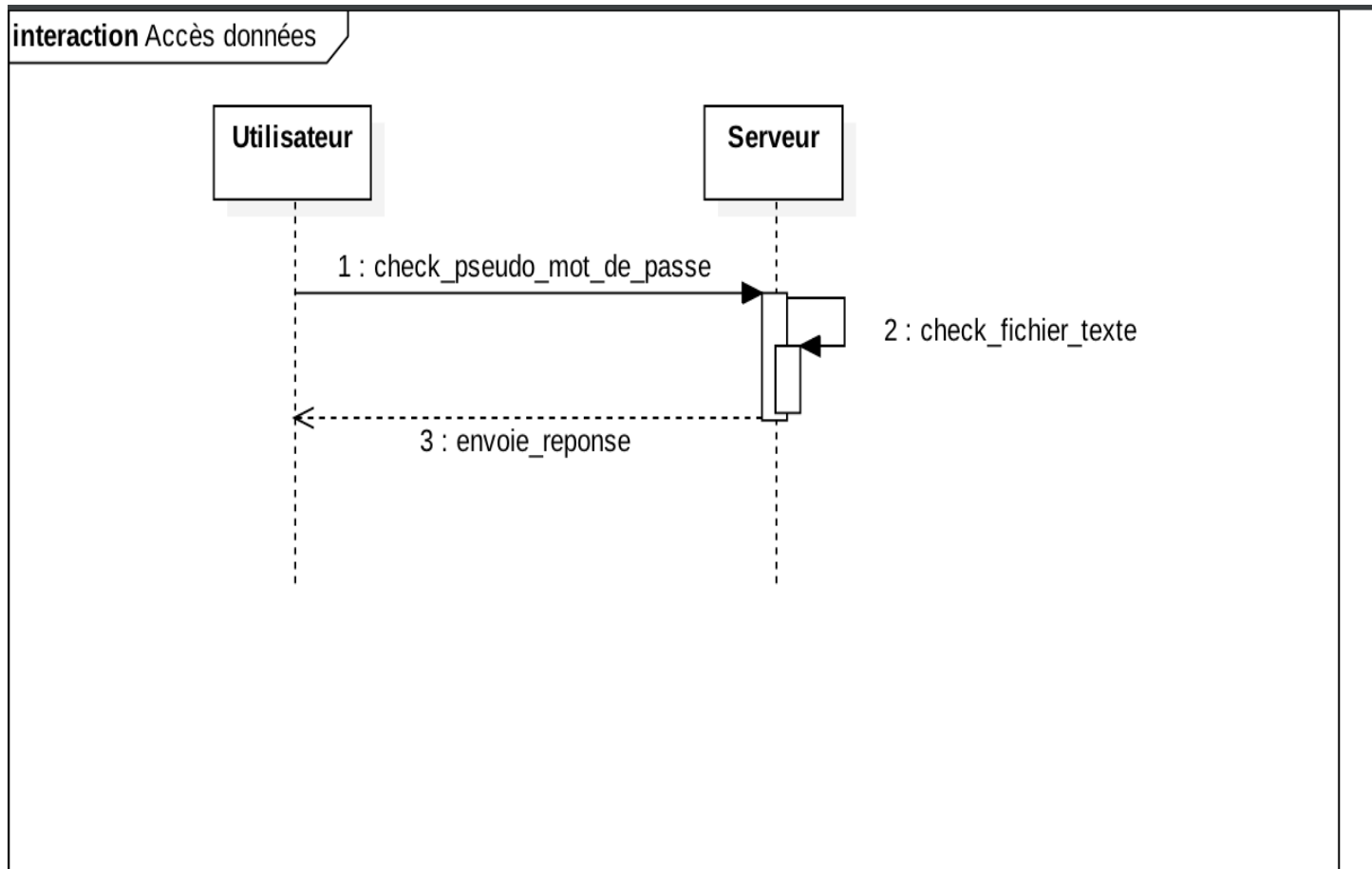


FIGURE 7 – Diagramme de sequence du Matchmaking.

8 Matchmaking

Lorsqu'un utilisateur veut jouer celui-ci va être repris par la classe Matchmaking. Ainsi, le matchmaking se chargera de placer chaque utilisateurs dans le bon lobby, en fonction du mode de jeu choisit par celui-ci, si le lobby est plein, le matchmaking va automatiquement créer un nouveau salon. Un lobby est considéré comme plein lorsque la limite max pour ce lobby est atteinte. Il est limité à 4 utilisateurs pour le mode par équipe et le mode versus, pour ce qui est du mode chrono, un seul utilisateur suffit.

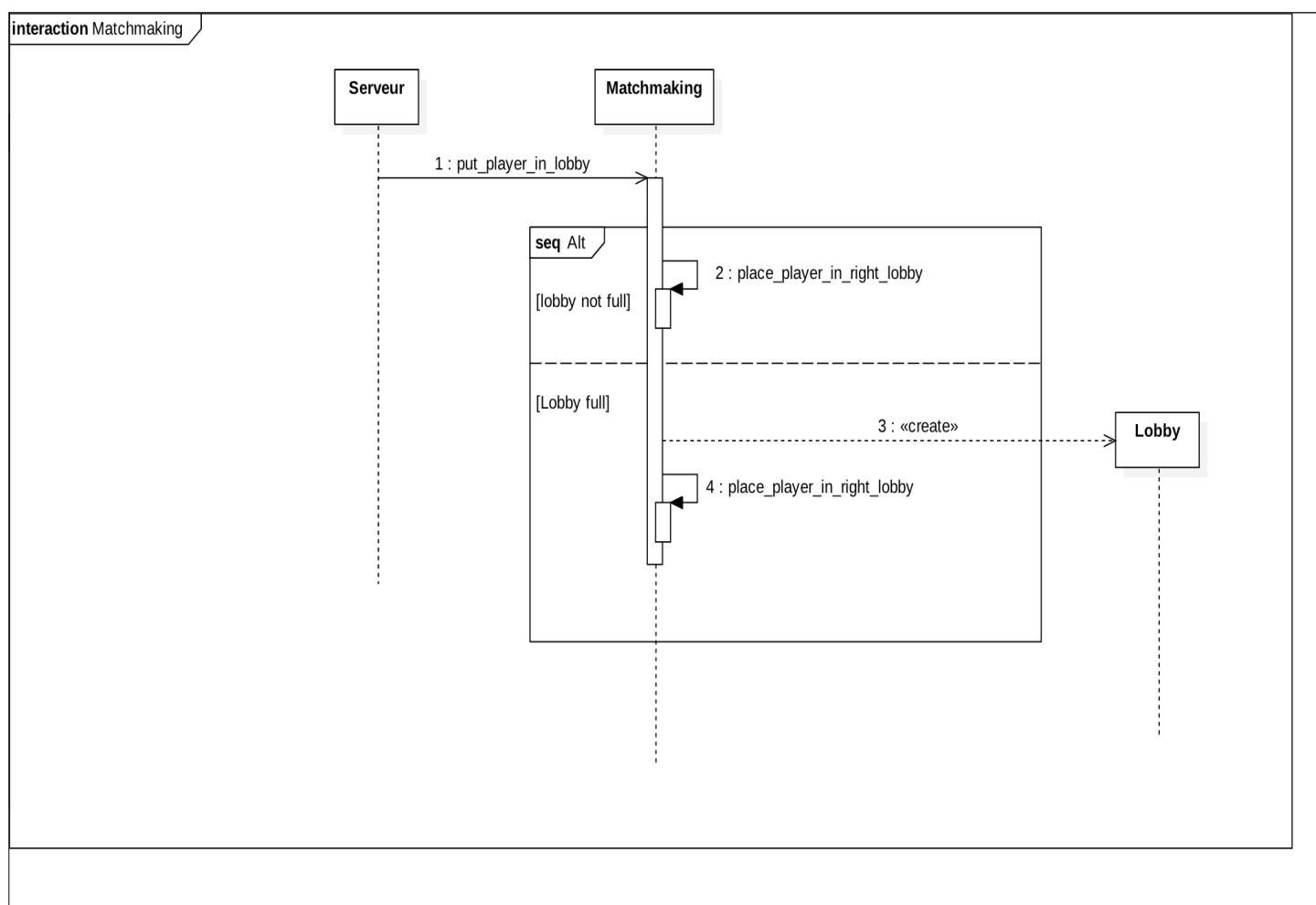


FIGURE 8 – Diagramme de sequence du Matchmaking.

9 Affichage console

La gestion de l’affichage a été faite de manière à représenter à tout moment les actions que le client décide de faire. De ce fait, on peut donc séparer l’affichage en 3 grosses parties.

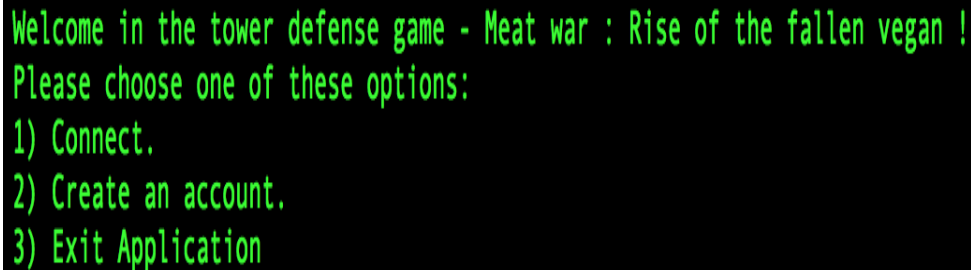
Inscription/connexion

le menu représentant les choix possibles(consulter la liste d’amis...)

L’affichage du jeu.

9.1 Inscription / connexion

La première partie consiste donc à l’inscription ou la connexion d’un utilisateur. Ainsi, L’utilisateur est invité à saisir un pseudo et un mot de passe dans le cas où il veut se connecter, et de manière similaire il est convié à saisir un nouveau pseudo et un nouveau mot de passe dans le cas où celui-ci veut s’inscrire.

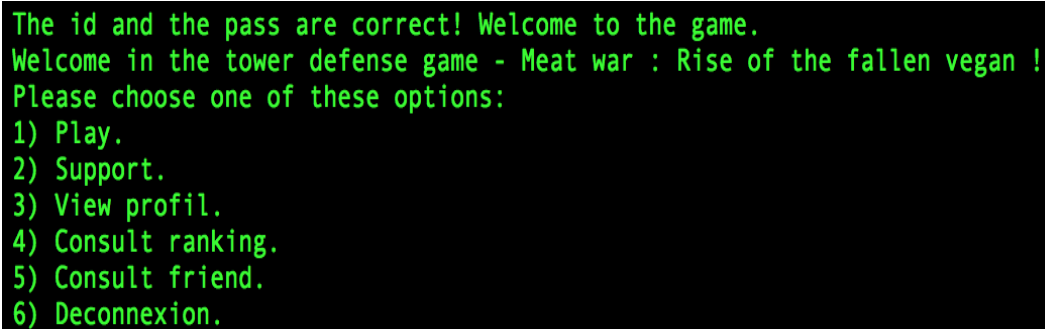


```
Welcome in the tower defense game - Meat war : Rise of the fallen vegan !  
Please choose one of these options:  
1) Connect.  
2) Create an account.  
3) Exit Application
```


9.2 Menu

Cette partie est donc accessible après que l'utilisateur s'est connecté, suite à une confirmation de la validité du compte réalisé par le serveur. Ensuite, 6 options sont mises à disposition du joueur ;

- Lancer une partie.
- Soutenir un joueur dans une partie.
- Visualisation du profil.
- Affichage du classement.
- Accéder au gestionnaire d'amis.
- Quitter.



```
The id and the pass are correct! Welcome to the game.  
Welcome in the tower defense game - Meat war : Rise of the fallen vegan !  
Please choose one of these options:  
1) Play.  
2) Support.  
3) View profil.  
4) Consult ranking.  
5) Consult friend.  
6) Deconnexion.  
█
```

De manière similaire l'accès au gestionnaire d'ami mène à plusieurs nouveaux choix :

- Ajout d'un ami.
- Suppression d'un ami.
- Accepter une demande d'ami.
- Consulter sa liste d'ami.
- Quitter.

L'ensemble de ces méthodes sont donc disponibles dans la classe Menu et dans la classe Friend.

9.3 Affichage du jeu

L’affichage en jeu est réparti par plusieurs objets. De manière générale, c’est la classe *grid* qui se charge de l’affichage de la carte mais aussi des différentes informations telles que :

- La vie des différents utilisateurs.
- Le score des utilisateurs.
- L’argent des utilisateurs.

D’autres part l’affichage des différents monstres(Végan) est délégué à la classe *Enemy*. Finalement, il ne reste plus que l’affichage des tourelles est c’est la classe *Tower* qui s’en charge.

10 Interface graphique

L'interface graphique a été réalisé en dernier lieu, se joignant ainsi au jeu déjà préconçu. Pour ce faire nous avons décidé de laisser le choix à l'utilisateur, celui-ci peut donc choisir entre l'interface graphique et l'ancienne partie sur terminal. Sur les différentes bibliothèques disponibles, nous avons choisis "qt". De ce fait, plusieurs moyens ont été mis en place pour coller le plus possible au thème(Meat VS Vegan). Premièrement plusieurs fond d'écran sont disponible et appliqué dans le jeu. Ensuite, La carte en jeu à été totalement revu et transformé en une carte plus adapté au thème. Puis les options disponibles dans le terminal ont été retravaillées pour qu'elles soient évidemment plus visuelles et plus faciles d'utilisation.

Pour l'implémentions de cette partie graphique nous avons dérivés des classes existantes de la bibliothèque "qt". De cette dérivation, nous avons donc apporté nos propres modifications personnalisées. De cette manière, nous avons une bonne structure orientée-objet et il suffit d'instancier l'objet là ou il le faut. Pour finir nous avons décidé de cacher les fenêtres au lieu de les détruire à chaque fois.

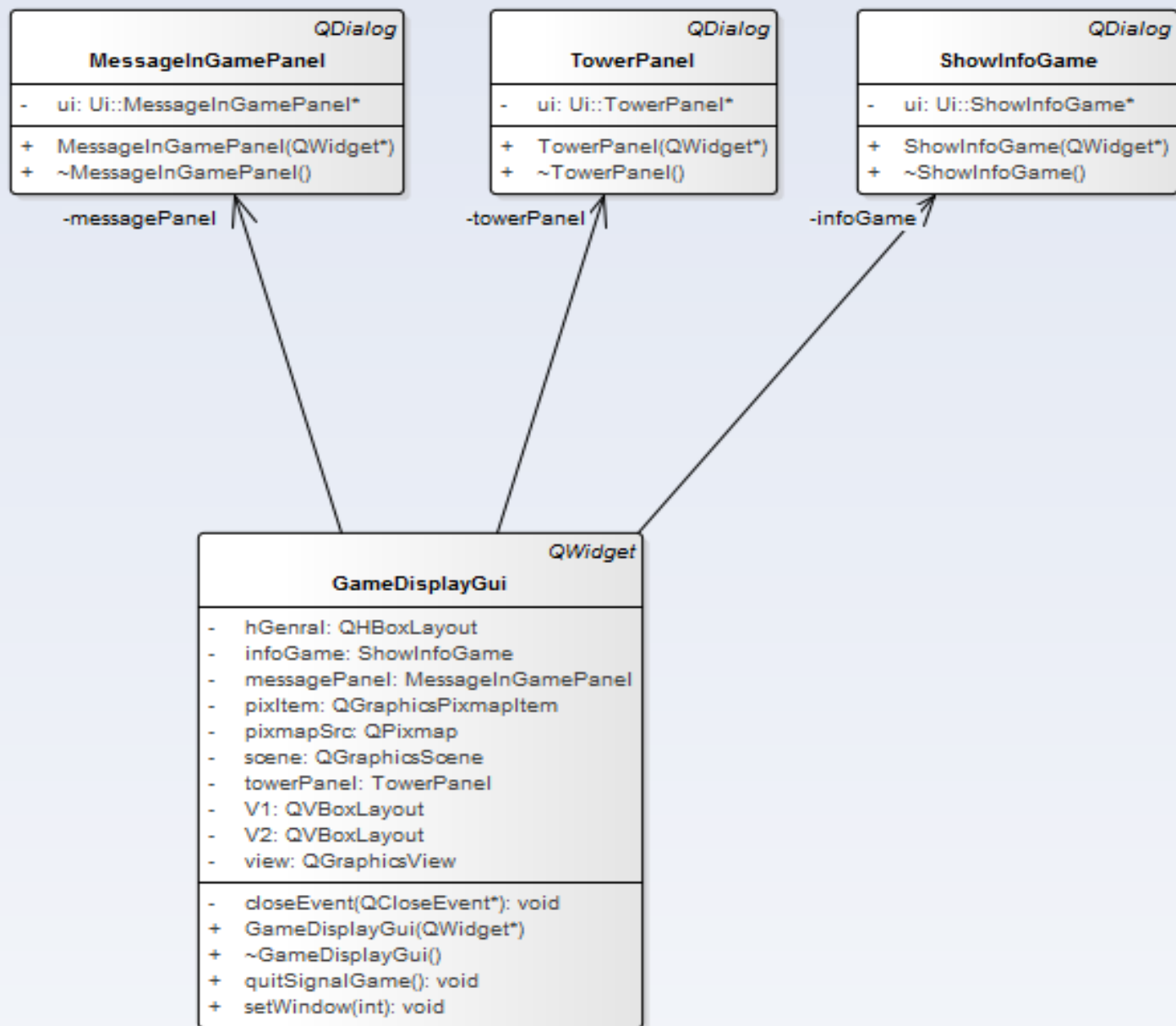
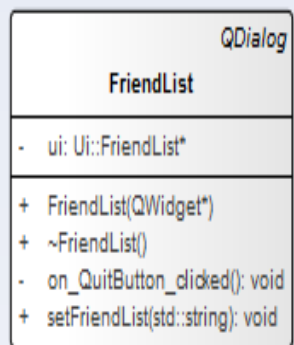
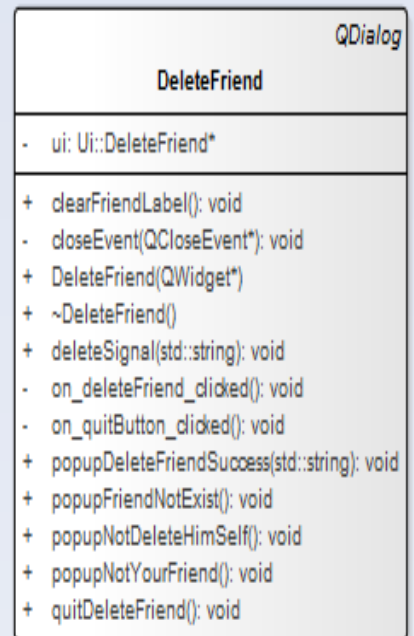
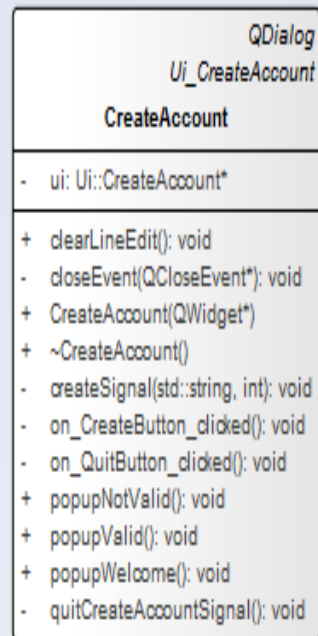
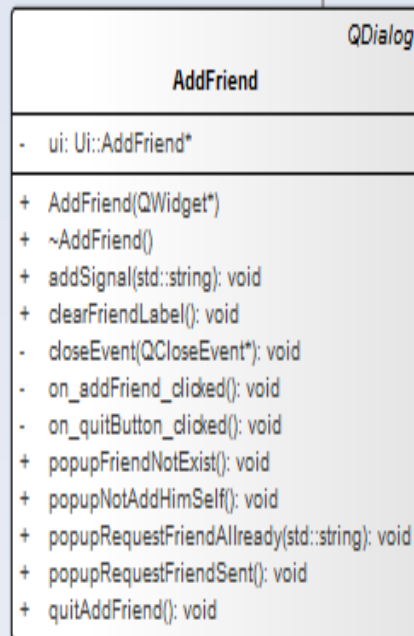
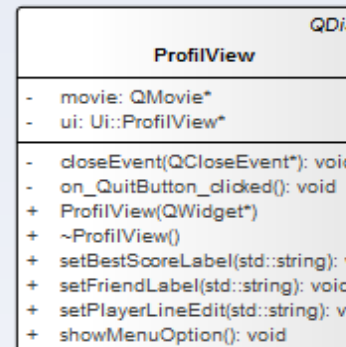
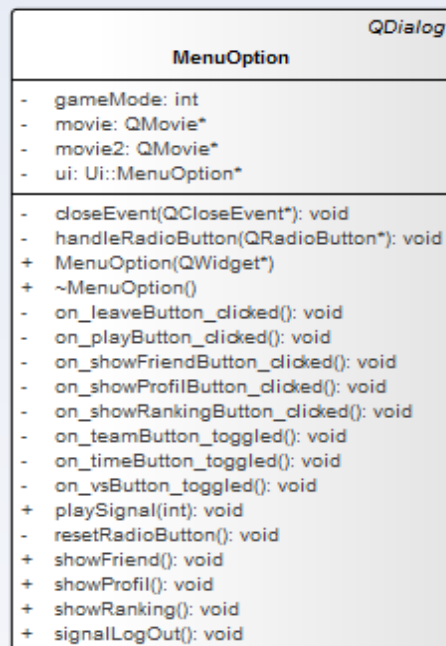
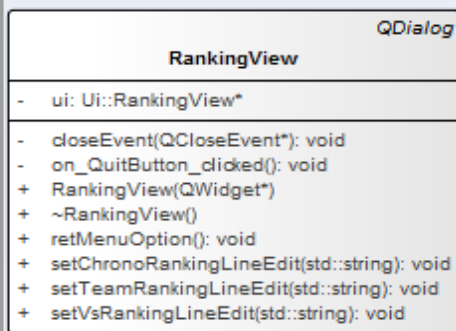
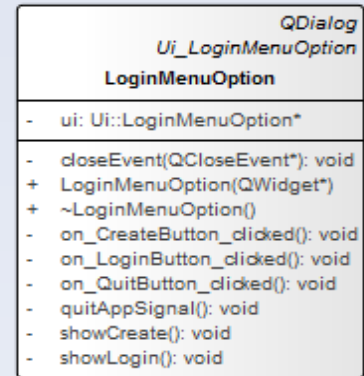
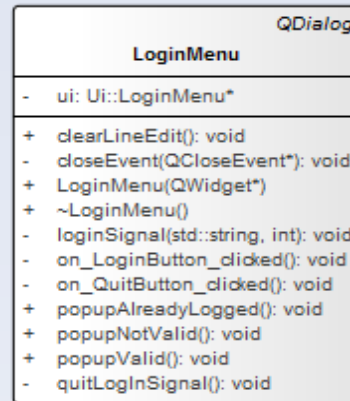
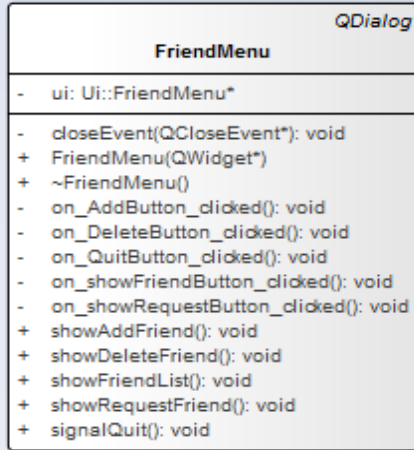


FIGURE 9 – Diagramme de classe partie GUI





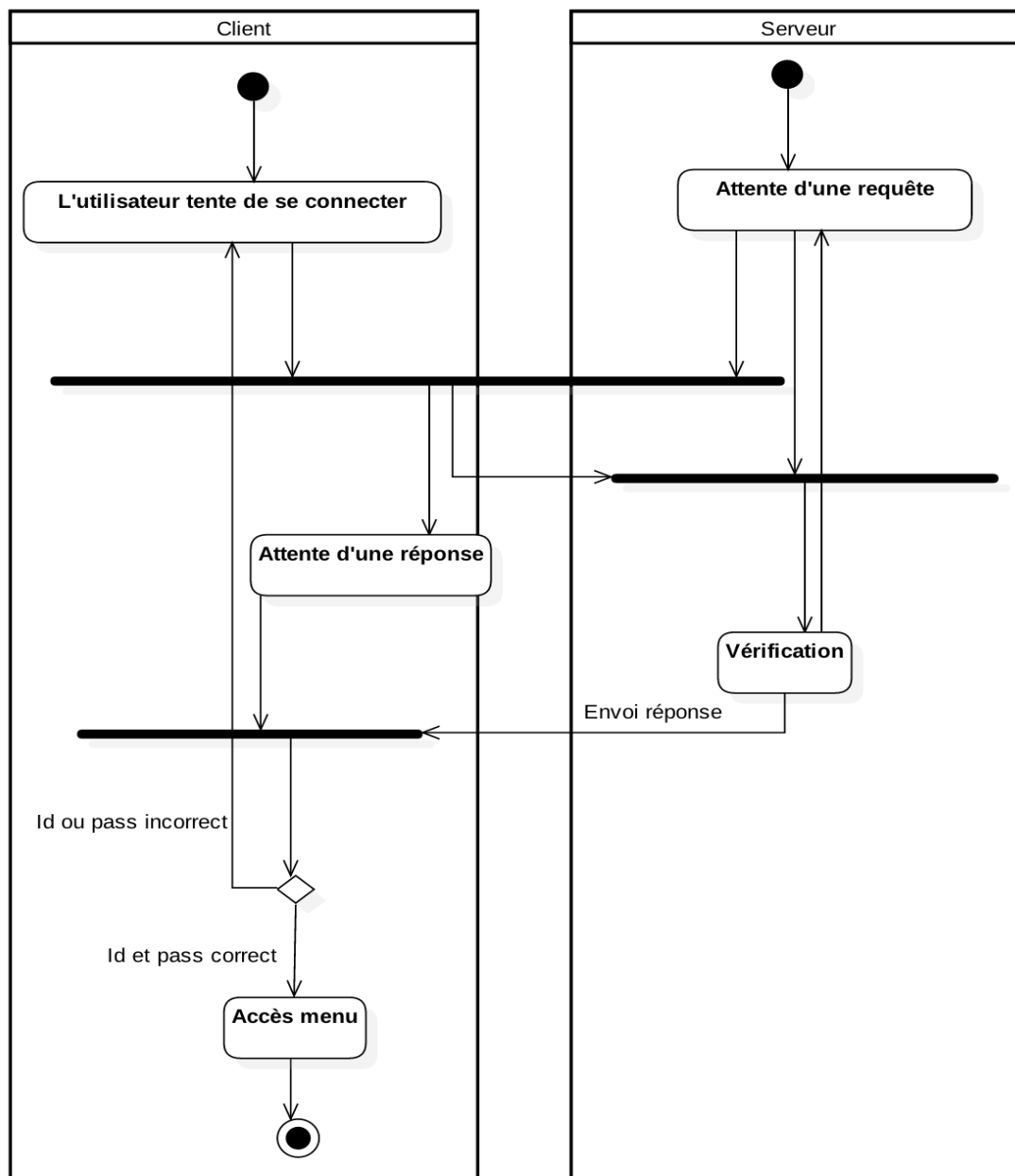


FIGURE 10 – Diagramme d'activité se connecter

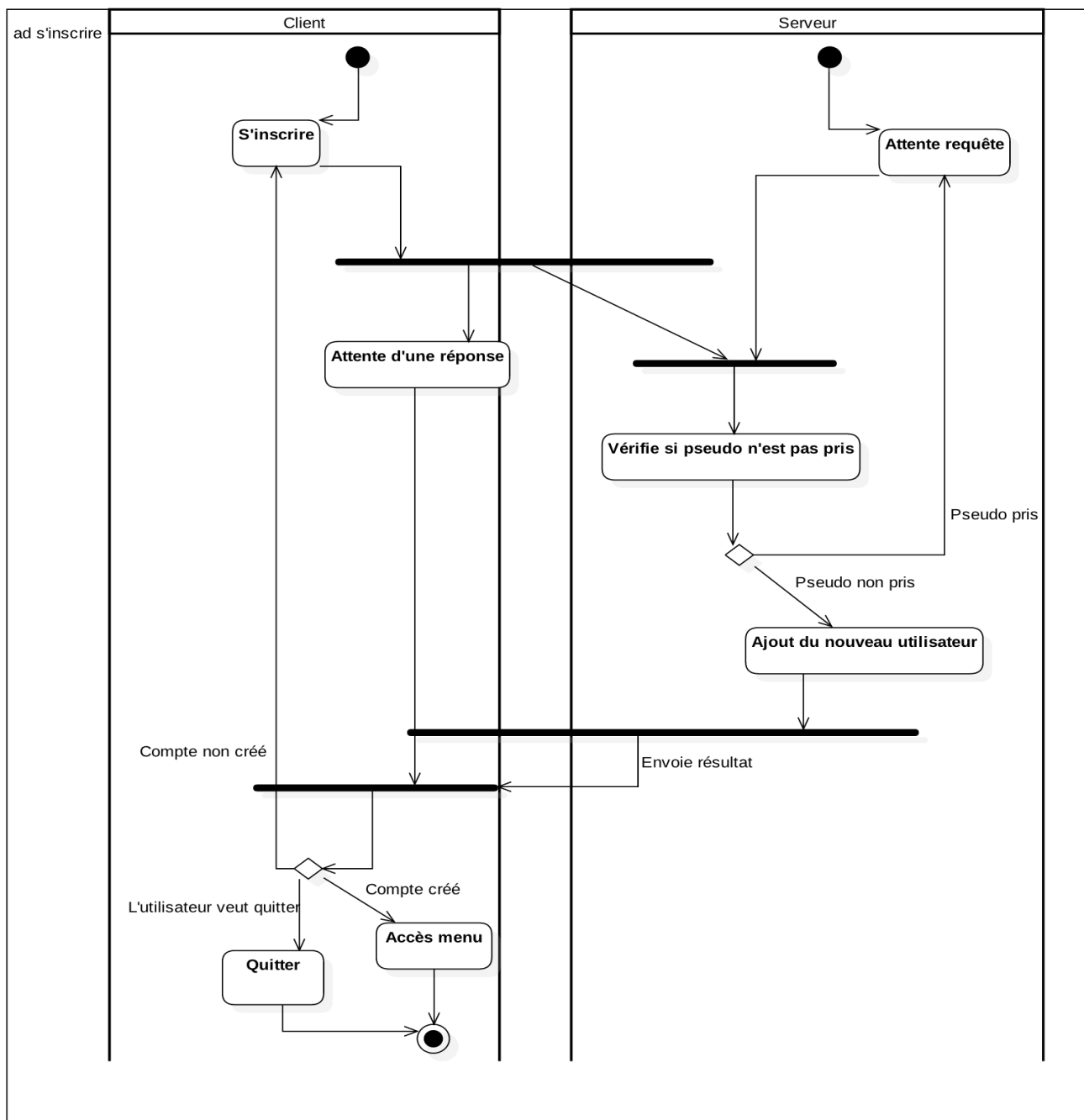


FIGURE 11 – Diagramme d'activité inscription

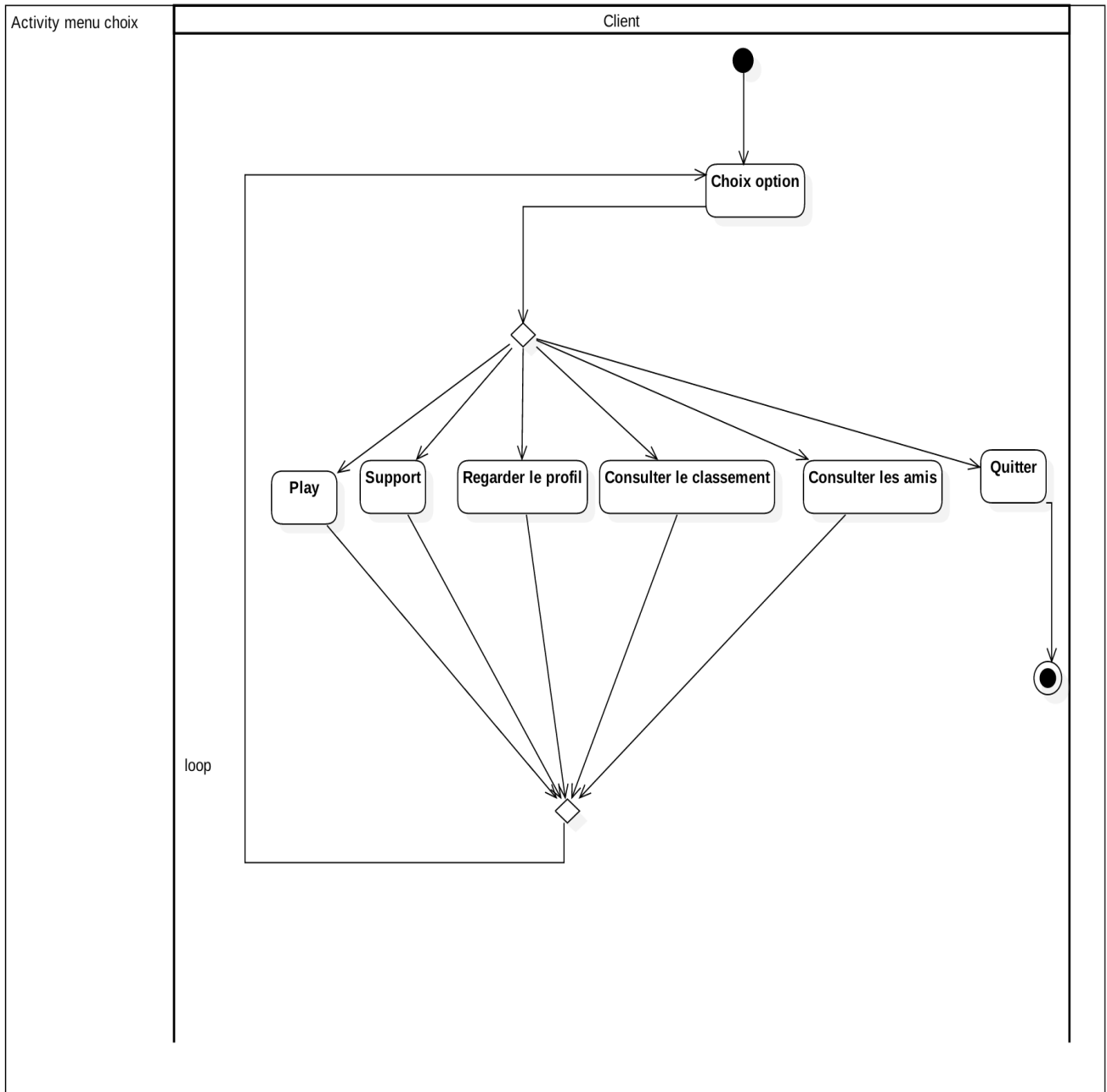


FIGURE 12 – Diagramme d’activité sur le menu

Index

Tower Defense, 3
client, 3

abandon, 11
argent, 3

base, 11

classement, 6
client, 3, 12, 21, 24
compte, 3, 6, 12
consulter son profil, 6

ennemi, 4, 11

joueur, 3, 6, 11, 12

liste d'amis, 6

map, 3

menu, 6, 11
mode chrono, 11

partie, 6, 11, 12
partie classique, 11
pseudo, 23, 26

salon, 12
serveur, 3, 11, 12, 24, 27
supporter, 6

thread, 21
tourelle, 28
tours, 3
Tower Defense, 3, 6, 11

utilisateur, 3, 6, 11, 12, 20, 21, 23–29
utilisateurs, 21, 23, 25