

Touché Coulé: SRD  
*Projet d'informatique 2 (INFO-F209)*

Islam BENAÏM SAHTANE RGHIF, Romain CROUGHS, Noah DEBUS,  
Alessandro DORIGO, Gabriel GOLDSZTAJN, Manon GORTZ, Pawel JADCZUK,  
Philippe NYABYENDA, et Lucas VAN PRAAG

Département d'Informatique, Faculté des Sciences, Université Libre de Bruxelles

Année 2023-2024

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	But du projet . . . . .	2
1.2	Glossaire . . . . .	3
1.3	Historique . . . . .	3
<b>2</b>	<b>Besoins utilisateurs</b>	<b>4</b>
2.1	Connexion . . . . .	4
2.2	Menu principal . . . . .	5
2.2.1	Lancement d'une partie . . . . .	5
2.2.2	Rejoindre une partie . . . . .	5
2.2.3	Gérer sa liste d'ami . . . . .	5
2.2.4	Chat . . . . .	6
2.3	En partie . . . . .	6
<b>3</b>	<b>Besoins système</b>	<b>7</b>
3.1	Connexion . . . . .	7
3.2	Match making . . . . .	8
3.3	Partie . . . . .	8
<b>4</b>	<b>Design et fonctionnement du Système</b>	<b>9</b>
4.1	Design du système . . . . .	9
4.1.1	Connexion et Inscription . . . . .	10
4.1.2	Gestion des comptes . . . . .	10
4.1.3	Menu principal . . . . .	11
4.1.4	Profil . . . . .	11
4.1.5	Matchmaking . . . . .	11
4.1.6	Chat . . . . .	12
4.1.7	Jeu . . . . .	12
4.1.8	Vue globale . . . . .	13
4.2	Fonctionnement du système . . . . .	15
4.2.1	Inscription . . . . .	15
4.2.2	Connexion . . . . .	16
4.2.3	Chat . . . . .	16
4.2.4	Matchmaking . . . . .	17
4.2.5	Lobby . . . . .	18
4.2.6	Boucle de jeu . . . . .	19

# Chapitre 1

## Introduction

### 1.1 But du projet

Ce projet a pour objectif de créer une expérience immersive où deux joueurs s'affrontent dans le célèbre jeu de société "Bataille navale". Chacun des joueurs dispose d'un plateau de jeu unique, dissimulé à son adversaire, sur lequel il place stratégiquement ses navires. Le déroulement du jeu se fait tour à tour, avec les joueurs tentant de localiser et de couler les navires adverses en lançant des attaques sur des coordonnées spécifiques. Dans le jeu originel, lorsqu'une attaque atteint un navire, la case correspondante est signalée par un point rouge ; autrement, elle est marquée d'un point blanc. La victoire est remportée par le joueur qui parvient à couler tous les navires adverses.

Deux modes de jeu sont disponibles : le mode classique et le mode commandant.

Mode classique :

Les flottes des deux joueurs se composent de cinq navires de tailles variées : 2x1, 3x1, 3x1, 4x1, et 5x1. Ces navires peuvent être orientés de 90 degrés, mais leur placement en diagonale n'est pas autorisé. De plus, deux navires ne peuvent pas être adjacents l'un à l'autre. Chaque tour permet à un joueur de tirer un coup, mais tant que le tir touche un navire, le joueur peut continuer à tirer.

Mode commandant :

Dans le mode commandant, différentes factions sont introduites, chacune avec ses propres navires et compétences distinctes. Contrairement au mode classique, dans ce mode, deux navires peuvent être positionnés côte à côte. Au début de son tour, chaque joueur reçoit des points d'énergie. Chaque action, comme le tir, consomme des points d'énergie. De nouvelles actions sont également disponibles, telles que l'utilisation d'un sonar révélant une zone de 3x3, le déclenchement d'un bombardement aérien ciblant une zone de 4x1 et bien d'autres. Si un joueur n'utilise pas tous ses points d'énergie au cours d'un tour, il peut les conserver pour le tour suivant, offrant ainsi une flexibilité tactique pour planifier des actions stratégiques plus complexes.

Un mode "extra" supporter est également présent. Dans ce mode, l'utilisateur observe une partie jouée par un ami.

En dehors d'une partie, un joueur a la possibilité de discuter avec ses amis et de gérer sa liste d'amis.

## 1.2 Glossaire

- Matchmaking : Mécanisme de mise en relation des joueurs afin de créer une partie en ligne.
- Plateau (Board) : Deux matrices de jeu, qui sont associées à deux joueurs.
- Lobby : Interface/lieu où l'on regroupe des joueurs avant de démarrer une partie.
- Player (Joueur) : Membre de la partie qui communique avec le jeu suite aux actions (input) de l'utilisateur
- User (Utilisateur) : Client qui est connecté en réseau et lié à compte.

## 1.3 Historique

Version	Nom	Modifications	Date
0.1	Manon Gortz	Introduction	3/12/23
0.2	Manon Gortz	Besoins utilisateur et squelette besoins système	3/12/23
0.3	Manon Gortz	Ajout des besoins non fonctionnels système	7/12/23
0.4	Manon Gortz	Description du design du système	7/12/23
0.5	Manon Gortz	Version incomplète du diagramme de classe	7/12/23
0.6	Manon Gortz	Squelette fonctionnement système	7/12/23
1.1	Pawel Jadczyk	Modification de la structure du fichier	13/12/23
1.2	Pawel Jadczyk	Correction et retouche du fichier	13/12/23
1.3	Pawel Jadczyk / Philippe Nyabyenda	Ajout diagrammes de séquence (chp4)	14/12/23
1.4	Manon Gortz / Noah Debus	Modification finale du diagramme de classe (version soft et version full)	15/12/23
1.5	Pawel Jadczyk / Romain Croughs / Noah Debus	Restructuration du fichier et annotation des nouveaux sous-diagrammes	15/12/23
1.6	Pawel Jadczyk / Romain Croughs	Glossaire	15/12/23

## Chapitre 2

# Besoins utilisateurs

### 2.1 Connexion

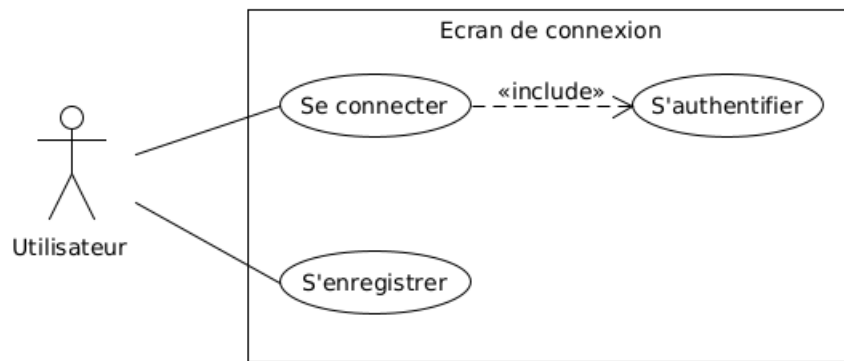


FIGURE 2.1 – Connexion

Lorsque le jeu est lancé, une fenêtre de connexion s'affiche, offrant à l'utilisateur la possibilité de se connecter à son compte. Si l'utilisateur ne dispose pas encore d'un compte, il a la possibilité d'en créer un. En cas de saisie de données incorrectes, le programme informe l'utilisateur de l'erreur. Une fois l'authentification réussie, l'utilisateur est dirigé vers le menu principal du jeu.

## 2.2 Menu principal

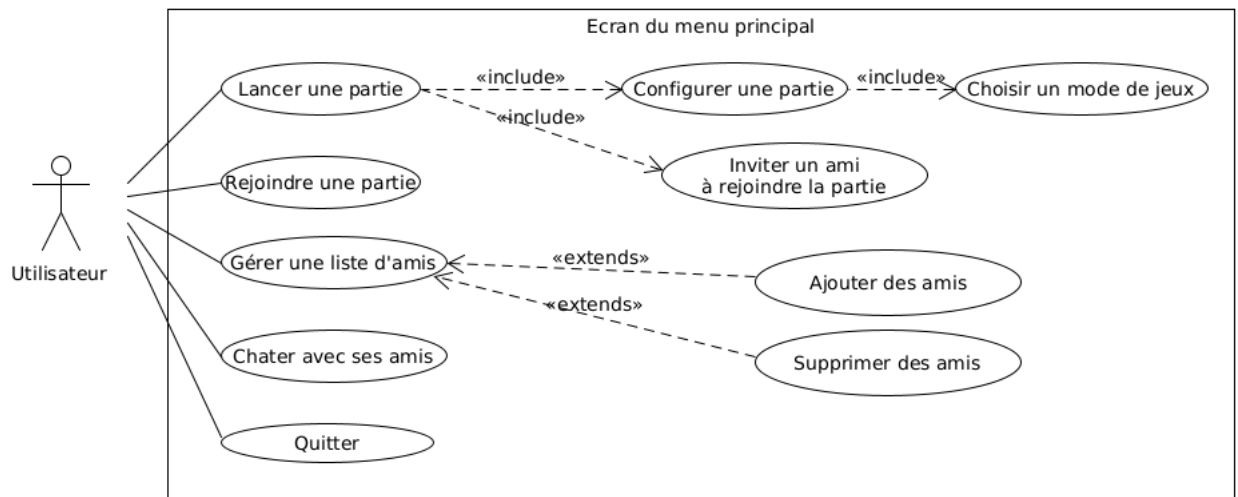


FIGURE 2.2 – Menu principal

### 2.2.1 Lancement d'une partie

Lors du lancement d'une partie le joueur doit simplement fixer les règles en choisissant :

- Le type de partie : **classique** ou **commandant**
- Le temps d'une partie : au maximum **15 minutes**
- Le temps par tour d'un joueur : au maximum **30 secondes**

Après quoi il sera redirigé vers un lobby, auquel il pourra inviter des joueurs, qui seront tous en mode spectateur s'ils acceptent l'invitation. Un joueur peut aussi rejoindre le lobby sans recevoir d'invitation. Si au moins un joueur à rejoint la partie, le joueur principale peut le fixer en tant que second joueur et lancer la partie. Mais avant que la partie ne puisse commencer, il faudra placer tous les bateaux disponibles sur le board, après quoi la partie se lance.

### 2.2.2 Rejoindre une partie

Lorsqu'un utilisateur est invité par un ami a rejoindre une partie, il a la possibilité de rejoindre la partie en acceptant l'invitation, mais il peut aussi refuser ou encore ignorer l'invitation. Si jamais le joueur est déconnecté, un message d'erreur est affiché pour informer qu'il n'est pas possible d'inviter le joueur. Un joueur a aussi la possibilité de rejoindre une partie sans invitation à condition que la partie soit au stade de lobby.

### 2.2.3 Gérer sa liste d'ami

Chaque utilisateur possède une liste d'amis qu'il peut modifier. Il a la possibilité de :

- ajouter un ami a sa liste
- retirer un ami de sa liste

### 2.2.4 Chat

Si le joueur a des amis en ligne dans sa liste d'amis, il a la possibilité de communiquer avec eux via un chat. Si un ami n'est pas connecté, ses messages cesseront d'être transmis, et le programme lui présentera un message l'informant de cette situation.

## 2.3 En partie

Le jeu doit être conçu pour offrir une interface intuitive, facilitant ainsi les actions du joueur. Dans un premier temps, les deux joueurs sont invités à placer les bateaux sur le board. Un message d'erreur est affiché lorsque le bateau est placé sur une mauvaise case et auquel cas le joueur est invité à essayer une autre position. Le même procédé sera utilisé pour les tirs.

Une fois tout les bateaux placés sur le board, le timer commence et la game débute. Pour rendre l'expérience en jeu plus agréable et plus interactive, le temps de jeu ne doit pas être trop long.

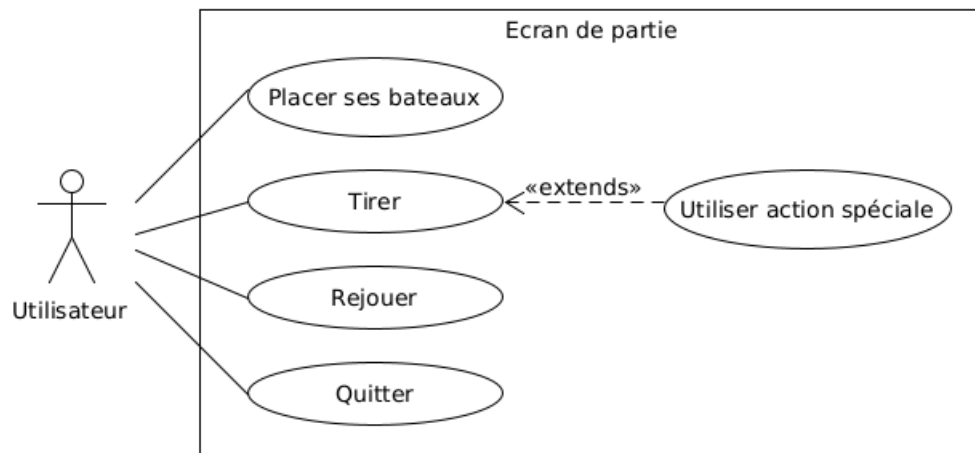


FIGURE 2.3 – En partie

## Chapitre 3

# Besoins système

### 3.1 Connexion

Le système doit pouvoir gérer la création de nouveaux comptes en vérifiant au préalable les données fournies. Une fois validé, les données sont sauvegardées dans le serveur.

Pour se connecter, le système vérifie l'authenticité des données dans le serveur.

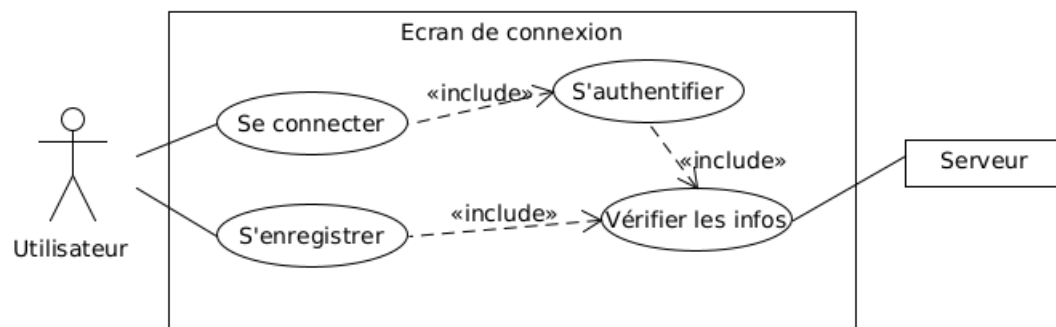


FIGURE 3.1 – Connexion



## 3.2 Match making

Durant le matchmaking, l'interface du lobby est affichée. On peut à ce moment inviter de nouveaux joueurs. Le lobby est joignable par d'autres joueurs sans nécessairement avoir envoyé d'invitations. Le système s'occupera de les ajouter au lobby et les définiras en tant que spectateur. Il faudra alors en définir un en tant que second joueur. Une fois cela fait, on peut lancer la partie. Le système s'occupera de vérifier cette contrainte et de passer au lancement de la game.

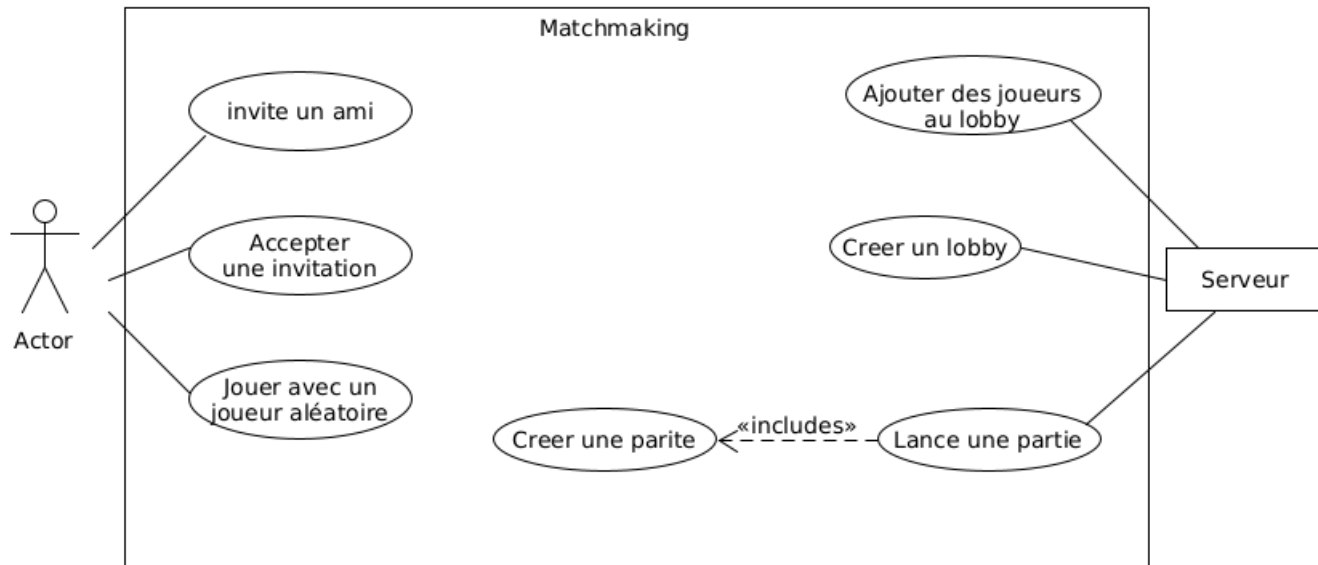


FIGURE 3.2 – Match making

## 3.3 Partie

Le système doit pouvoir permettre une action au tour par tour pour chaque joueur mais aussi vérifier la validité de ces actions. Il doit laisser le même joueur jouer à chaque fois qu'il touche un bateau ennemi.

Il permettra trois affichages distincts

- Le plateau avec les deux boards où le premier joueur ne voit pas les bateaux adverses.
- Le plateau avec les deux boards où le deuxième joueur ne voit pas les bateaux adverses.
- Le plateau avec les deux boards où le spectateur ne voit pas les bateaux adverses.

## Chapitre 4

# Design et fonctionnement du Système

### 4.1 Design du système

Le système est divisé en 7 entités :

- Connexion et Inscription (en rose)
- Gestion des comptes (en gris)
- Menu principal (en jaune)
- Profil (en bleu ciel)
- Matchmaking (en orange)
- Chat (en vert)
- Jeu (en bleu foncé)

Nous allons les détailler dans différents diagrammes. A noter que toutes ces entités vont hériter des mêmes classes abstraites. Cela est dû au fait d'utiliser le design MVC pour cette application. Chaque entité du système est donc composée de 3 composantes principales :

- La vue : Elle correspond à l'interface, elle permet d'afficher les informations et de recevoir les input de l'utilisateur qu'elle communique au controller.
- Le controller : Il gère les requêtes envoyées par la vue et les communique au serveur avec lequel il échange des informations. Il met le serveur à jour suite aux actions d'utilisateurs.
- Le serveur : il contient toutes les informations relatives aux utilisateurs et à une partie. Il est responsable d'envoyer les informations requises par le controller et permet de mettre les joueurs en relation pour une partie.

Nous utilisons également la paterne observer/subject afin que la vue soit automatiquement mise à jour des modifications apportées au serveur ainsi que des changements d'un menu à un autre. Les classes vue et controller héritent donc d'une classe abstraite observer et le serveur d'une classe abstraite sujet.

En partie, chaque client est connecté au même plateau, et à son propre côté. Le game serveur fait le lien entre les deux. Le GameController permet de mettre à jour le plateau de chaque client après une action.

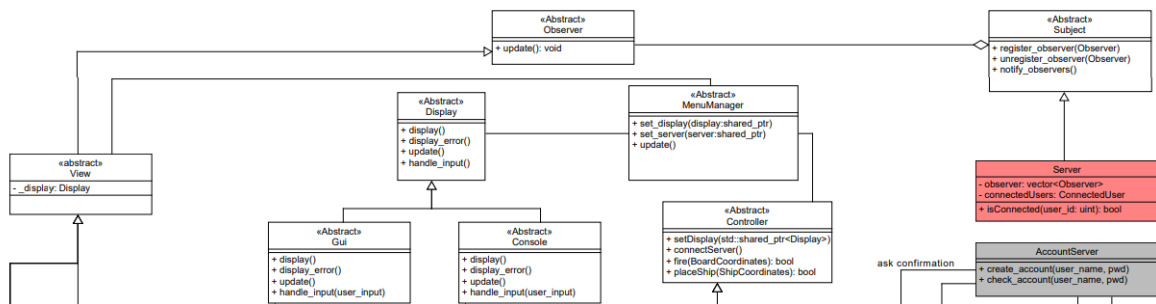


FIGURE 4.1 – Classes abstraites

### 4.1.1 Connexion et Inscription

Dès que le joueur lance le jeu, il a la possibilité de soit créer un compte, soit se connecter au compte qu'il a déjà. Il aura donc deux interfaces différentes pour ces deux actions.

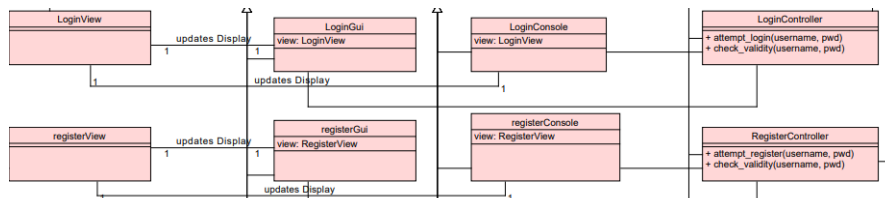


FIGURE 4.2 – Connexion et Inscription

### 4.1.2 Gestion des comptes

Le `AccountServer` permet de communiquer avec la base de donnée. Elle permet de produire des instances de la classe `Account`. Chaque compte a également une liste d'ami, qu'il peut gérer.

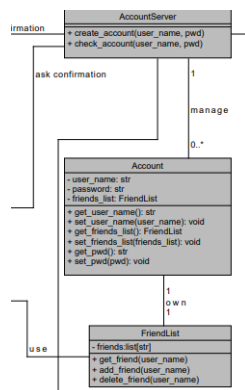


FIGURE 4.3 – Gestion des comptes

### 4.1.3 Menu principal

Le menu principal est celui qui permettra d'accéder aux autres écrans du jeu, tels que la création de partie, ou la modification de profil.

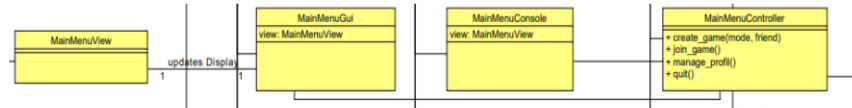


FIGURE 4.4 – Menu principal

### 4.1.4 Profil

Le joueur a accès à une interface qui lui permet de modifier les informations relatives à son compte.



FIGURE 4.5 – Profil

### 4.1.5 Matchmaking

La classe MatchMaker a une liste d'instances de PendingMatch, qui est une partie qui attend d'être lancée, car elle ne contient pas assez de joueurs.

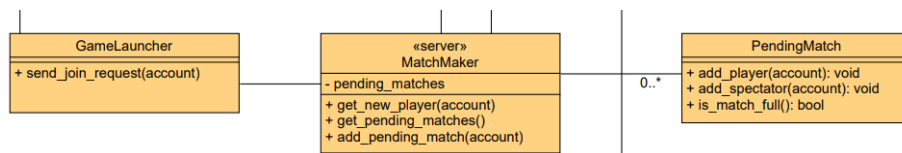


FIGURE 4.6 – Matchmaking

### 4.1.6 Chat

Le jeu offre aux joueurs un chat qui leur permet d'envoyer une chaine de caractère qui sera envoyée à tous les autres joueurs. Toute cette redirection de chaines de message se fait dans le **ChatServer**. De plus, le joueur a une liste d'ami, qui affiche tous ses amis, qu'ils soient connectés ou non.

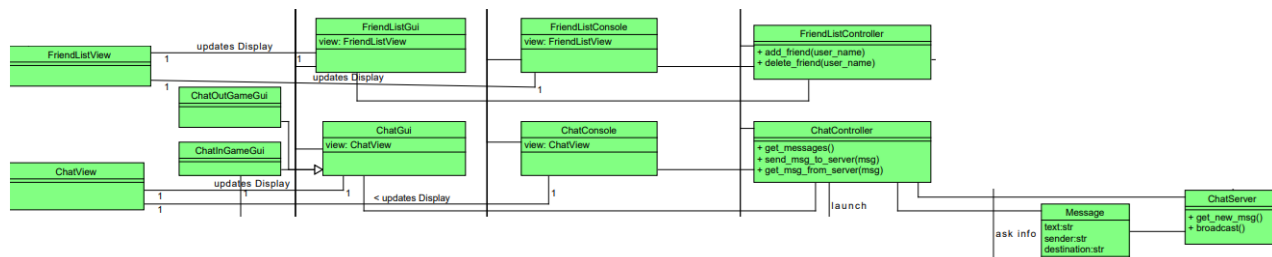


FIGURE 4.7 – Chat

#### 4.1.7 Jeu

La partie locale du jeu est conçue de la manière suivante : le joueur a une board local qui est une copie du board du serveur (avec les cases de l'adversaire qui sont masquées), un contrôleur qui permet de communiquer avec le serveur et un display qui permet d'afficher une partie à l'écran.

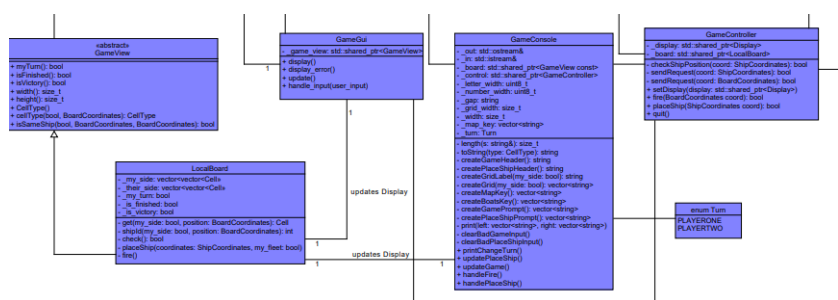


FIGURE 4.8 – Game affichage

En ce qui concerne la partie serveur d'une partie, nous avons une instance de Board qui contient les plateaux des deux joueurs, ainsi qu'une flotte qui appartient à ces deux joueurs. Le GameServeur est la classe qui s'occupe de la communication entre le Board et les joueurs.

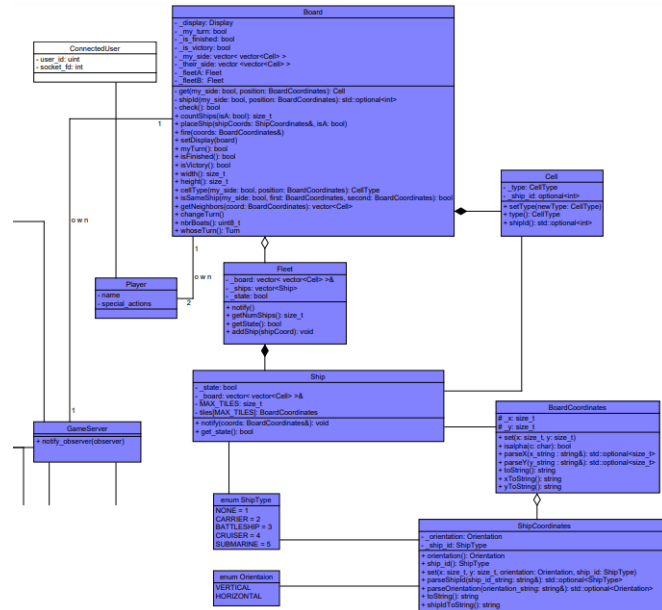


FIGURE 4.9 – Game serveur

#### 4.1.8 Vue globale

Pour aider à visualiser l'interaction entre les différents acteurs et avoir une vue globale. Nous avons fait un diagramme de classe qui reprend l'idée générale du système.

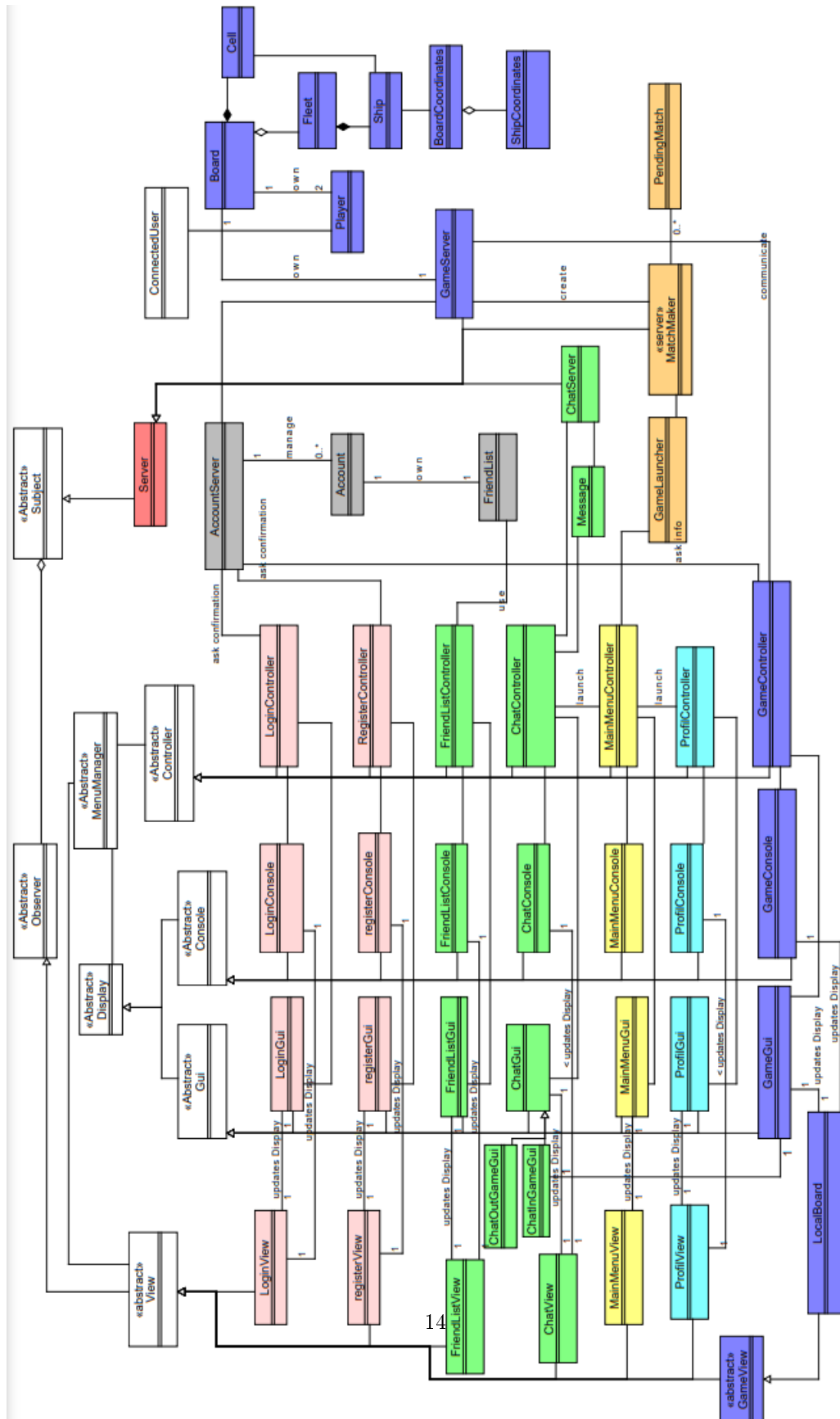


FIGURE 4.10 – Diagramme de classe

## 4.2 Fonctionnement du système

Cette section portera sur des diagrammes qui présenteront le fonctionnement de différentes parties du système.

### 4.2.1 Architecture de la communication client-serveur

Le client communique avec le serveur via une collection de requêtes HTTP. Techniquement, nous utilisons la bibliothèque `CppRestSDK` pour envoyer des requêtes HTTP.

Pour vérifier qu'un joueur soit bien connecté, le joueur dispose d'un token d'authentification qui est envoyé avec chaque requête. Ce token est généré lors de la connexion et est stocké dans le cache du client. Il n'est pas valide indéfiniment, et est régénéré à chaque nouvelle connexion. Chaque requête est interceptée par le serveur, qui vérifie la validité du token. Si le token est valide, la requête est traitée, sinon, une erreur est renvoyée. Ce token est envoyée pour toutes les requêtes qui nécessitent une authentification, comme par exemple, pour envoyer un message dans le chat, ou pour démarrer une partie.

De plus, bien que tout le code soit écrit en C++, nous avons décidé de nous inspirer des architectures REST utilisées dans les applications web. C'est pour cela que nous avons décidé que toutes les requêtes seraient envoyées en format JSON, et les réponses sont également envoyées avec ce format. Afin de pouvoir traiter le format JSON, nous utilisons la bibliothèque de Niels Lohmann, `nlohmann/json`.

### 4.2.2 Inscription

Si le joueur possède déjà un compte, il a la possibilité de se connecter directement. Le gestionnaire d'inscription (`registerController`) effectuera une première vérification des données saisies par l'utilisateur, notamment en s'assurant qu'aucun champ n'est laissé vide.

Une fois cette vérification effectuée, les données seront transmises au serveur via l'endpoint `api/register`. Ce dernier prendra comme argument la nom d'utilisateur et le mot de passe.

Ce dernier procédera ensuite à la vérification de la disponibilité du nom d'utilisateur. Afin de maximiser la sécurité, le mot de passe sera haché avant d'être stocké dans la base de données. La base de donnée contiendra alors le nom d'utilisateur et le mot de passe haché. Cependant, pour une recherche dans la base de donnée, il est bien plus efficace de donner un identifiant unique à chaque utilisateur. C'est pour cela que nous avons décidé de donner un identifiant unique à chaque utilisateur. Ce dernier sera généré automatiquement lors de l'inscription et est récupérable via l'endpoint `api/login/uid`.

Maintenant que tout s'est bien déroulé, l'utilisateur va automatiquement être connecté. C'est à dire que le token d'authentification sera généré et envoyé au client.

*Remarque :* Le token a deux fonctions : à savoir qui envoie le message, et savoir si la personne qui envoie le message est bien connecté. Par exemple, le token est particulièrement utile pour le chat, car il permet de savoir qui envoie le message, et si l'utilisateur est bien connecté (pour éviter que tout le monde puisse récupérer les message d'un autre utilisateur).



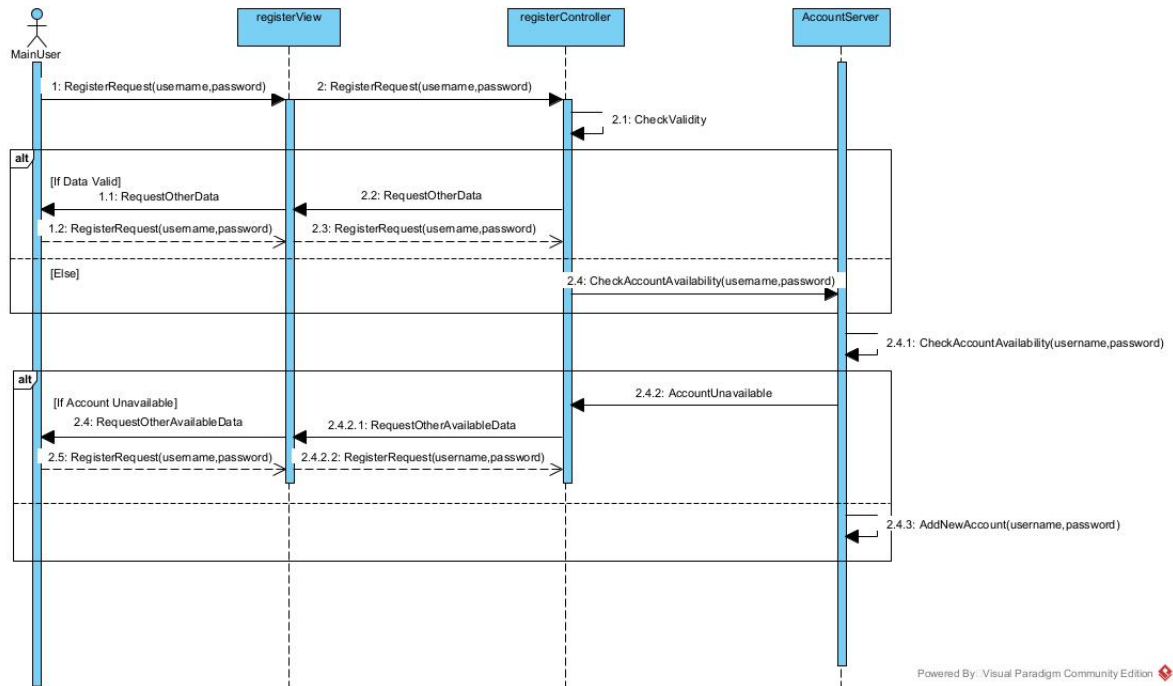


FIGURE 4.11 – Inscription

### 4.2.3 Connexion

La connexion est en quelques sortes similaire à l'inscription. L'utilisateur doit saisir son nom d'utilisateur et son mot de passe, et le gestionnaire de compte va vérifier si le compte est bien valide. Si c'est le cas, le serveur va générer un token d'authentification et l'envoyer au client. Dans le cas contraire, le client va recevoir un code d'erreur.

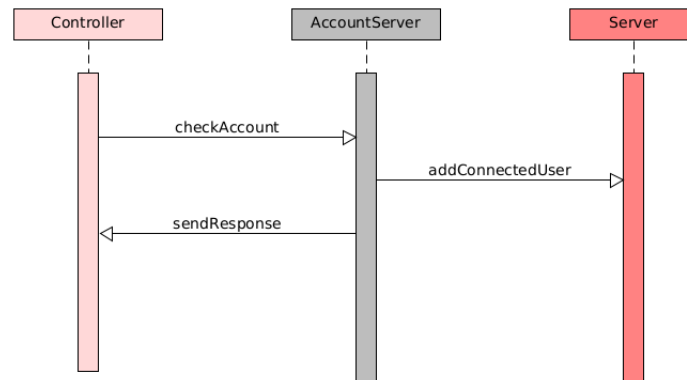


FIGURE 4.12 – Connexion

### 4.2.4 Chat

Le chat permet aux utilisateurs de communiquer les uns entre les autres. Il s'agit uniquement de *whispers* (messages privés), qui ne comprennent que des messages textuels. Le chat est complètement géré par le serveur, c'est-à-dire qu'aucun message n'est stocké par l'utilisateur.

L'utilisateur a alors simplement deux endpoint pour chatter : `api/chat/send` pour envoyer un message, et `api/chat/get` pour recevoir les messages. Lors de chaque requête, l'utilisateur envoie son token et le nom de son ami. Le serveur vérifie alors si l'utilisateur est bien connecté, et si l'ami existe bien. Si c'est le cas, le serveur peut alors effectuer toutes les manipulations nécessaires pour envoyer ou recevoir un message.

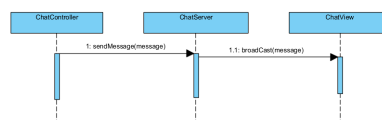


FIGURE 4.13 – Envoie d'un message

### 4.2.5 Profil

### 4.2.6 Matchmaking

Ce diagramme décrit la séquence d'actions réalisées lorsqu'un utilisateur veut créer/configurer une partie.

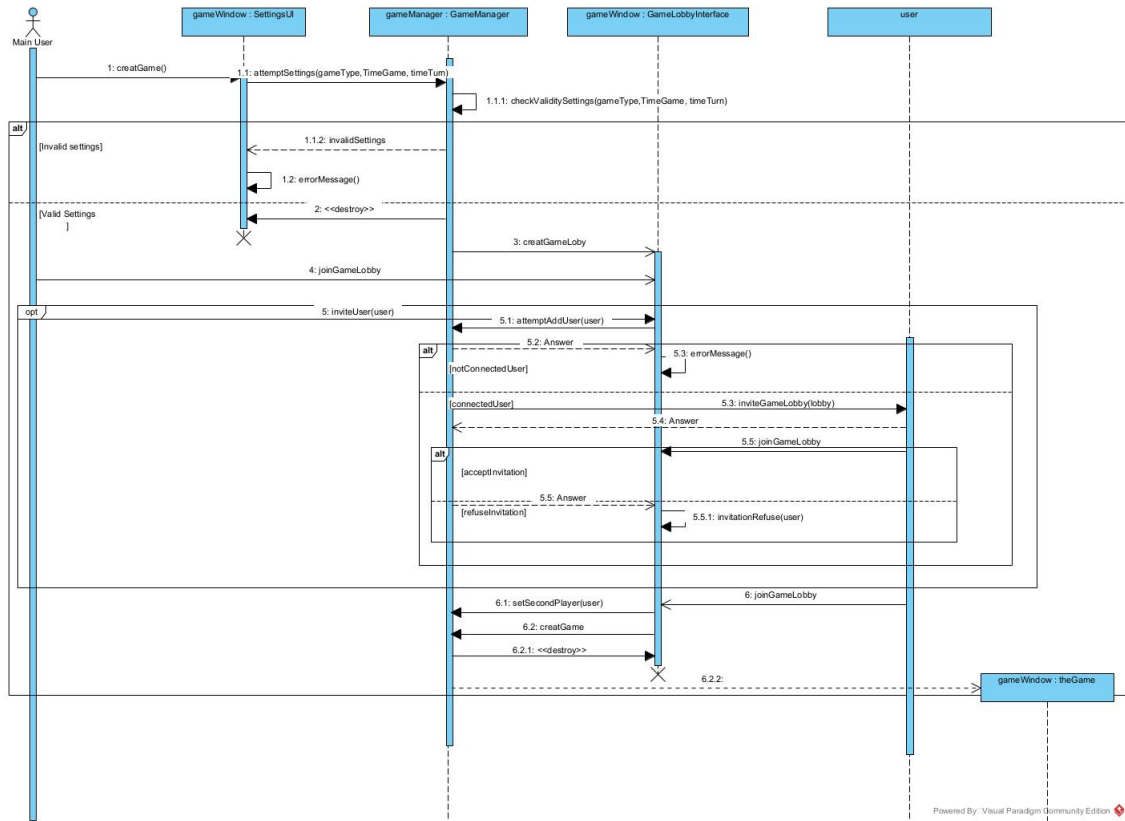


FIGURE 4.14 – Matchmaking : Création du lobby

## 4.2.7 Lobby

Ce diagramme décrit la séquence d'actions réalisés lorsqu'un utilisateur se trouve au stade de lobby et qu'il veut démarrer une partie. Le deuxième joueur a déjà été fixé, et les bateaux doivent être placés sur le plateau pour que la partie puisse se lancer. Mais avant de placer les bateaux, le plateau avec les deux boards est créé mais aussi 2 players. Un player est assigné à chaque user.

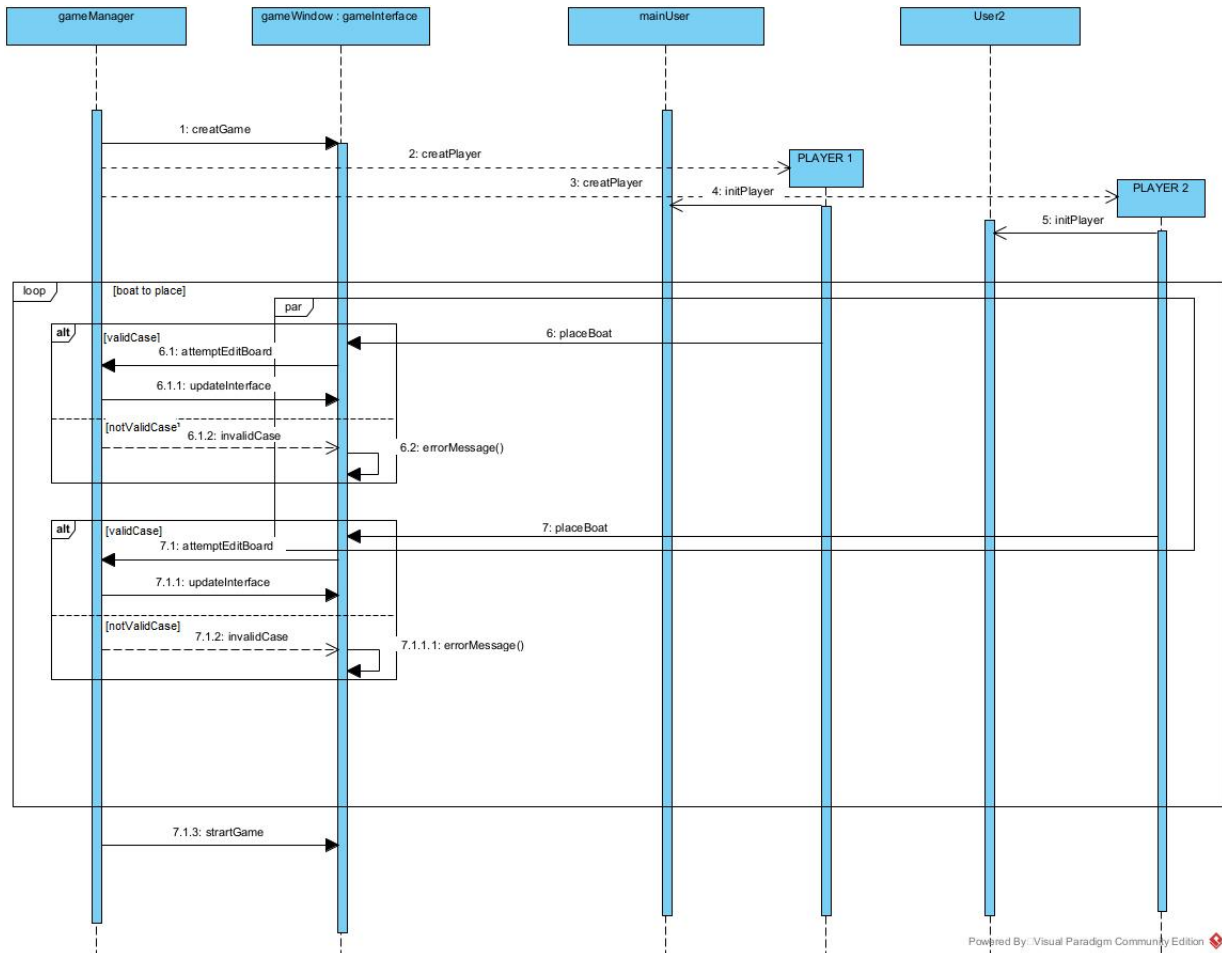


FIGURE 4.15 – Avant que la partie commence

### 4.2.8 Boucle de jeu

La déroulement de la partie se passe au tour par tour. A tour de rôle le user transmet son action mais l'action est gérée par player. Cela permet à une classe player d'avoir des points d'énergie, des actions et faire partie d'une faction. L'action est d'abord vérifiée pour savoir si la case ciblée est autorisée. Si c'est le cas le plateau est modifié et l'interface est mise à jour.

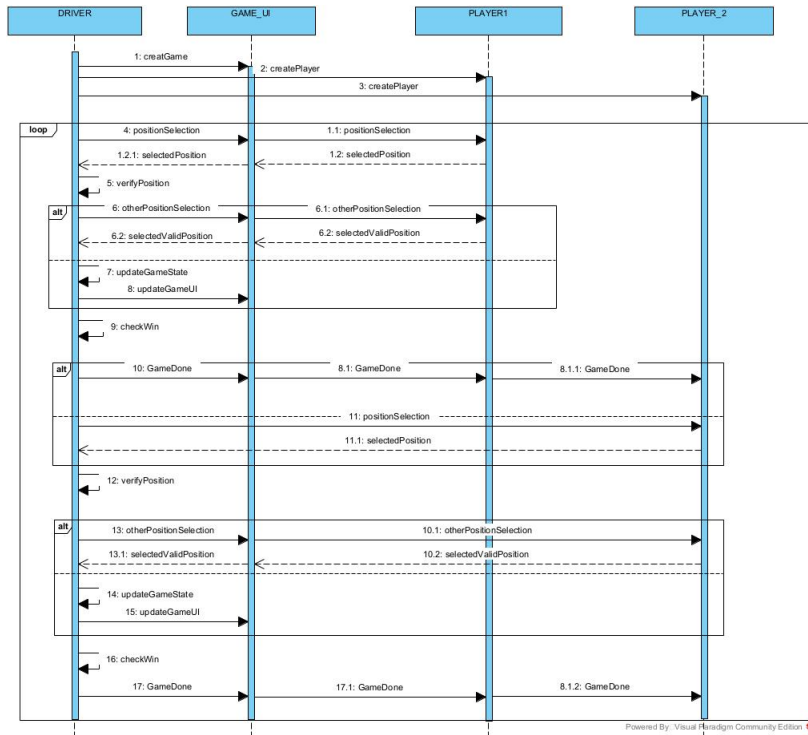


FIGURE 4.16 – Boucle de jeu