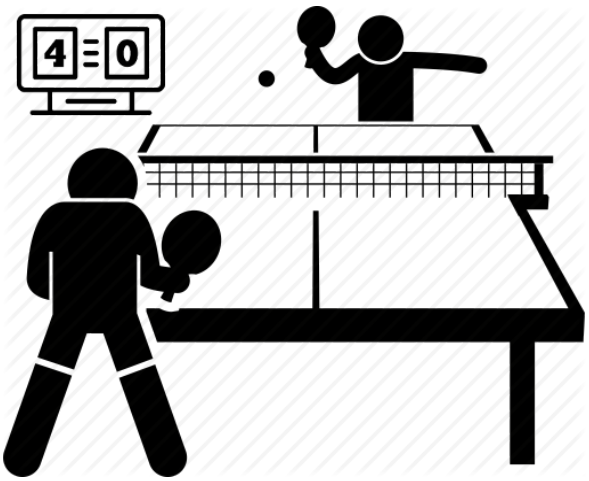
Dossier technique

Projet AREA

GEROUVILLE William **BRUNET Bastien**



BTS SN IR Lasalle Avignon 2021

SOMMAIRE

1. Introduction

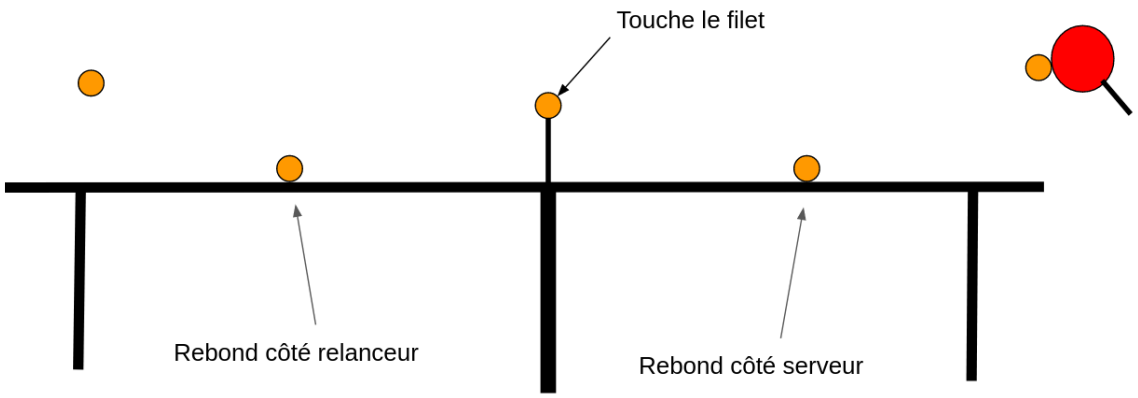
AREA est un système d'assistance à l'arbitrage qui permettra entre autres :
- De détecter avec précision un net lors d'un service
- De gérer les différentes parties d'une rencontre
- De faciliter l'arbitrage de chaque parties individuellement
- De permettre un affichage clair des informations liées à la rencontre au public

Les équipements mis en oeuvre seront :
- Des capteurs sur chaque côtés de la table
- Un capteur sur le filet
- Au moins un module mobile/tablette
- Un afficheur accompagné d'un système raspberry pi

L'arbitrage sera géré par un ou plusieurs arbitres qui pourront :
- Définir les noms des joueurs de chaque équipe
- Définir les noms de clubs s'affrontent
- Ajouter des points aux joueurs
- Accorder un temps mort pour un des deux camp

La phase de NET :

Un net est une action qui peut survenir lors de la phase de service au tennis de table. Pour réussir une service, le serveur doit taper la balle de façon à ce que la balle rebondisse de son côté puis du côté du joueur adverse, appelé relanceur. Si la balle rebondit du côté du serveur puis touche le filet pour enfin rebondir du côté relanceur il y a net et le service est à remettre.



2. Présentation générale

2.1. Expression du besoin

Actuellement il n'existe pas de système électronique permettant la détection de "net" communiquant ce qui rend cette dernière difficilement gérable par un arbitre. De plus, lors de compétitions par équipe de tennis de table, il est parfois difficile pour le public de suivre le déroulement de plusieurs parties en simultané.

Le ping pong club Sorgues souhaite donc dans ce cadre développer un système communiquant afin de :

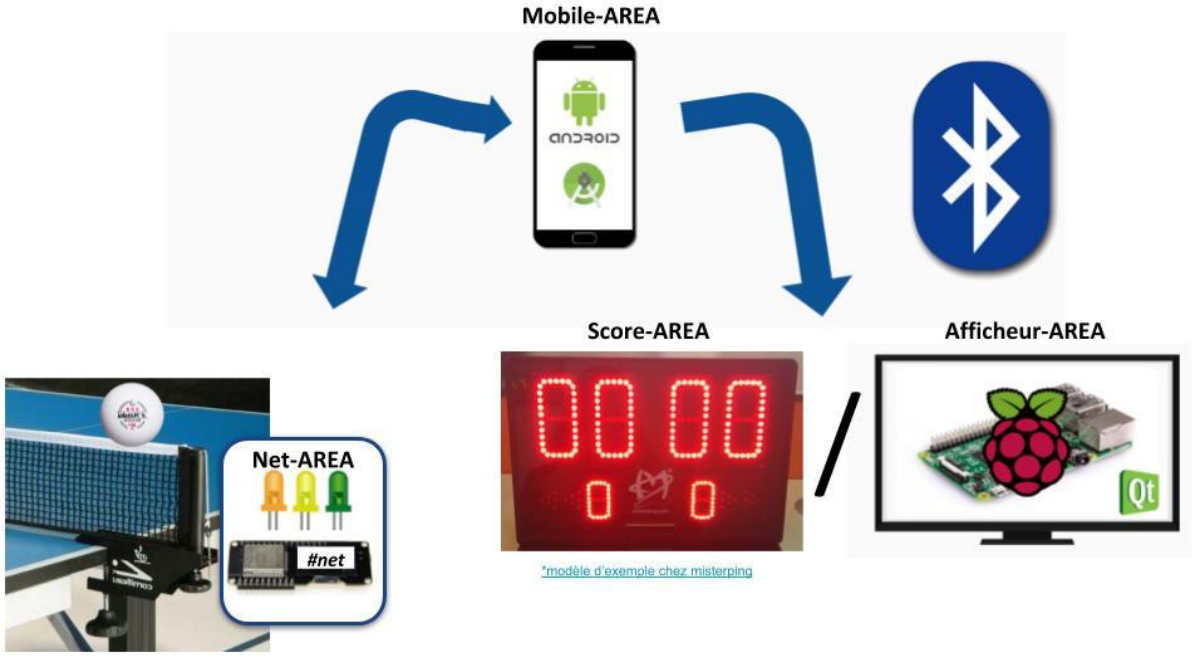
- Faciliter l'arbitrage en ajoutant la détection d'un "net" lors d'une phase de service
- Pouvoir gérer simplement le score d'une partie
- Faciliter la saisie des informations relatives à une rencontre (nom des joueurs, numéro de licence, nom du club et le nombre de manches gagnantes pour cette rencontre)
- Visualiser le déroulement d'une ou plusieurs parties

2.2. Présentation du projet

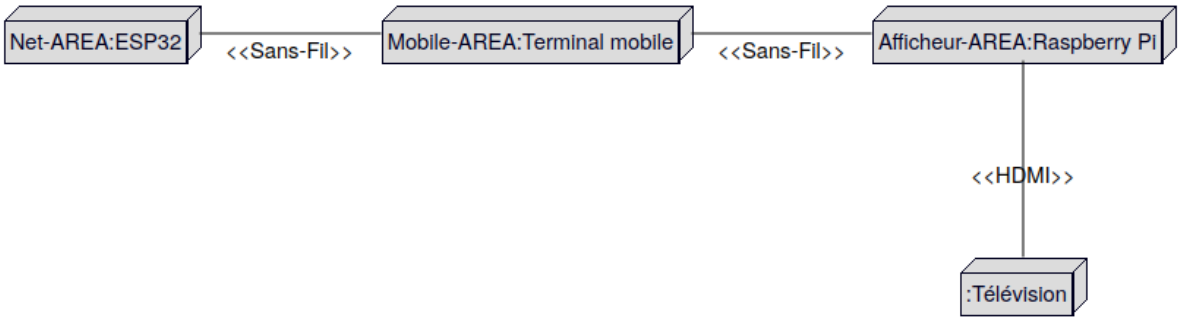
Il s'agit de réaliser un système d'assistance à l'arbitrage lors d'une partie de tennis de table en compétition qui permettra :

- De détecter la présence d'un "net" et de le signaler
- De gérer le score d'une partie de tennis de table a partir d'un terminal mobile
- De visualiser le déroulement d'une ou plusieurs parties sur un écran de télévision

2.3. Architecture du système



2.4. Diagramme de déploiement



Les modules NET, Mobile et Afficheur communiquent entre eux en bluetooth, le module afficheur communique via HDMI avec la télévision.

2.5. Répartition des tâches

Étudiant 1 (EC) : Module Net-AREA

- Les capteurs sont installés et fonctionnels
- L'impact de balles est détecté (côté relanceur et serveur 137 cm x 152 cm)
- L'impact sur le filet est identifié
- La transmission sans fil est fonctionnelle
- Les données sont transmises

Étudiant 3 (BRUNET Bastien IR) : Module Mobile-AREA

- L'édition des informations d'une rencontre (nom des joueurs, numéro de licence, nom du club et le nombre de manches gagnantes pour cette rencontre) est possible
- Les informations d'une rencontre sont sauvegardées dans une base de données
- Le lancement et la gestion d'une partie est possible
- La gestion du score d'une partie par l'arbitre est fonctionnelle
- L'affichage d'une détection de "net" lors d'un service est réalisé
- La liaison Bluetooth est fonctionnelle et les connexions des modules sont affichées
- Les informations d'une partie sont transmises aux modules d'affichage
- L'application mobile est déployée

Étudiant 4 (GEROUVILLE William IR) : Module Afficheur-AREA

- Le système d'exploitation est installé et fonctionnel
- L'écran est configuré en mode "kiosque"
- Le déroulement d'une partie est fonctionnel
- Les données d'une partie sont affichées en temps réel
- La liaison sans fil est opérationnelle

3. Présentation personnelle : Gerouville William

3.1 Rappel du besoin initial

AREA est un système qui doit faciliter l'arbitrage d'une partie de sport de duel tel que l'escrime, tennis de table, basket, volley-ball. Dans ce projet, le système ciblera plus spécifiquement le tennis de table en ajoutant une détection de "net" lors d'un service.

3.2. Organisation

Pour l'organisation du projet et la communication entre tous les membres du groupe, nous avons utilisé :

- Subversion qui est un logiciel libre de gestion de versions hébergé sur le site RiouxSVN pour l'ensemble du code source du projet.
- L'espace de stockage commun Google Drive pour tous les documents ressources.
- Le site internet Beesbusy afin de gérer l'avancement du projet et de générer un gantt

3.3. Objectifs

L'objectif de l'afficheur AREA est d'afficher en temps réel les informations d'une à deux parties simultanément tout en affichant l'historique de la rencontre.
Les données à afficher sont récupérées d'un module mobile connecté en Bluetooth afin de recevoir les données liées à l'arbitrage en temps réel.

3.4. Outils de développement

Désignation	Caractéristiques
Système d'exploitation du poste de développement	Linux 4.8.0 (Ubuntu 16.04)
Outil de planification	BeesBusy
Atelier de génie logiciel	Bouml 7.8
Logiciel de gestion de version	Subversion
Générateur de documentation	Doxygen
Environnement de développement	Qt Creator 3.5.1 (c++)

Qt Creator

Qt Creator est un environnement de développement intégré (IDE) multi-plateformes facilitant la programmation en C++, JavaScript et QML d'interfaces graphiques.

Il intègre un outil de développement d'interface graphique: Qt Designer qui sera ici utilisé pour la mise en place de l'IHM.

BOUML

Bouml est un designer de diagrammes en Langage de Modélisation Unifié (UML) multilingue et multiplateforme qui permet la génération de code ainsi que la rétro-ingénierie.
Les diagrammes générés suivent la norme UML 2.0.

Subversion

Apache subversion (abrégé subversion ou svn) est un logiciel de gestion de versions, distribué sous licence Apache s'appuyant sur le principe du dépôt centralisé et unique.

Doxygen

Doxygen est un générateur de documentation sous licence libre. Il tient compte de la syntaxe du langage dans lequel est écrit le code source, ainsi que des commentaires s'ils sont écrits dans un format particulier.

3.5. Répartition des tâches

Le projet étant décomposé en plusieurs itérations (développement itératif) voici comment il s'organise.

Tâche à réaliser	Priorité	Itération
Prévoir la possibilité d'afficher plusieurs match	moyenne	1
Afficher les informations de la rencontre (points, participants...)	haute	2
Assurer la communication avec le module Mobile_AREA	haute	2
Afficher les fautes en temps réel	haute	2
Afficher des informations sur la prochaine rencontre	moyenne	2
Gérer les temps morts (demandeur, temps restant, ...)	moyenne	3
Configurer l'écran en mode kiosque	basse	3
Afficher les informations sur les rencontres précédentes (lors des temps morts, des pauses, ...)	basse	3
Finaliser l'apparence de l'application	basse	3

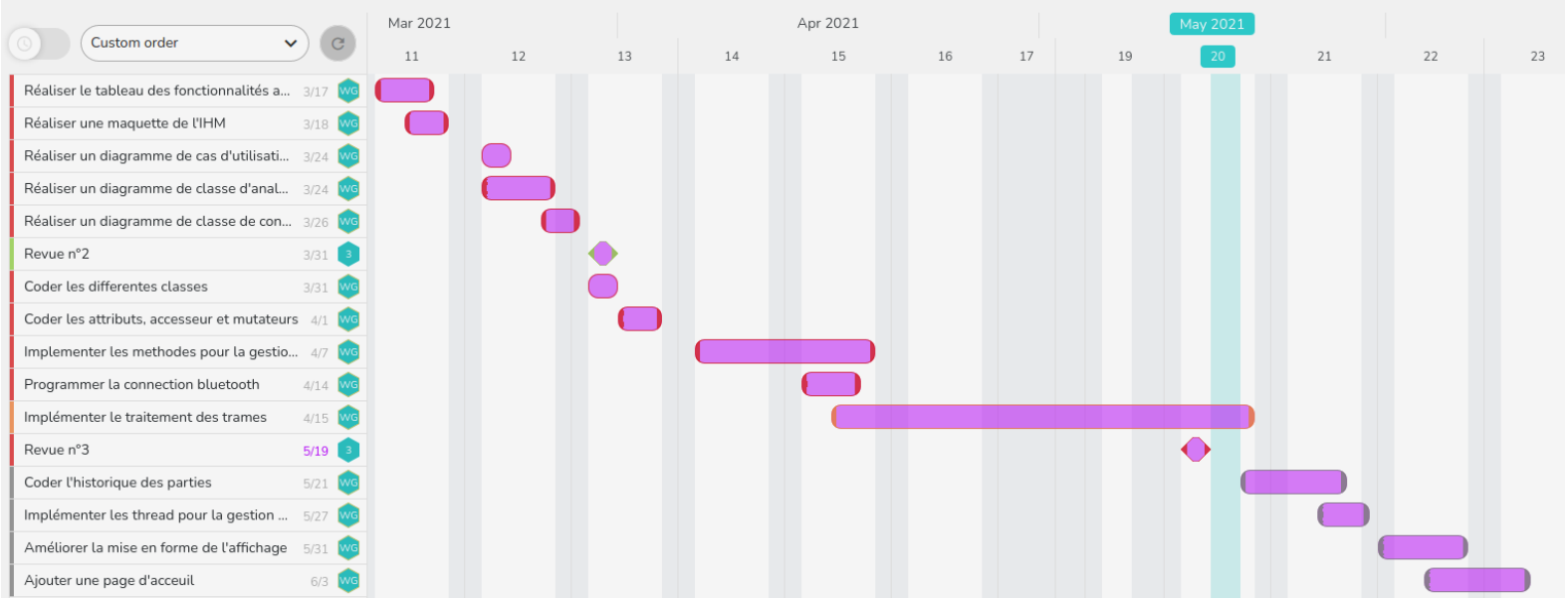
La première itération se concentre sur l'organisation, la mise en place de l'ihm et la prise en main des différents outils de développement.

La deuxième itération contient donc la programmation des fonctionnalités "coeur" de l'application, avec en priorité haute : la communication avec un module mobile et l'affichage des informations principales.

L'itération finale portera sur le peaufinage de l'affichage et quelques détails visuels.

3.6. Planification

Diagramme de gantt



En accord avec la répartition des tâches, la première itération se concentrait sur la mise en place de l'ihm et du squelette du code, la résolution des premiers problèmes se présentant et la réalisation de diagrammes.

Les semaines suivantes étaient basées sur le développement des fonctionnalités de l'application puis la période la plus longue concernait la communication entre modules, étant la fonctionnalité principale de l'afficheur c'est la tâche la plus longue du projet.

La fin du projet se concentrait sur la finalisation de l'affichage et la lisibilité des informations sur l'affichage.

3.7. Diagramme de cas d'utilisation

Le spectateur doit pouvoir visualiser les informations en **temps réel**, l'écran étant configuré en mode **kiosque** toutes les informations doivent être transmises via la liaison bluetooth.
Ici "visualiser les scores" insinue la visualisation de toute les informations, ceci inclut :

- Le score de chaque joueurs dans une partie
- Le chronomètre de la partie
- Le nombre de manches remportées par chaque joueurs d'une partie
- Les noms et prénoms des participants d'une partie
- Les mêmes informations pour une deuxième partie simultanément
- L'historique des parties / déroulement de la rencontre

3.8. Diagrammes de classes

La classe principale est la classe **Rencontre**, elle permet le lien entre l'IHM et la gestion des données dans le programme, elle possède des méthodes permettant de créer une partie constituée elle-même de 2 à 4 joueurs.

La classe **Partie** joue aussi un rôle important dans l'organisation puisqu'elle permet de gérer l'affichage de multiple parties simultanément, l'attribut "affichage partie" est un entier qui permet de définir si la partie n'est pas affichée à l'écran (si il est égal a 0) si la partie est actuellement affichée à gauche(s'il est égal à 1) ou bien si elle est affiché à droite (s'il est égal à 2).

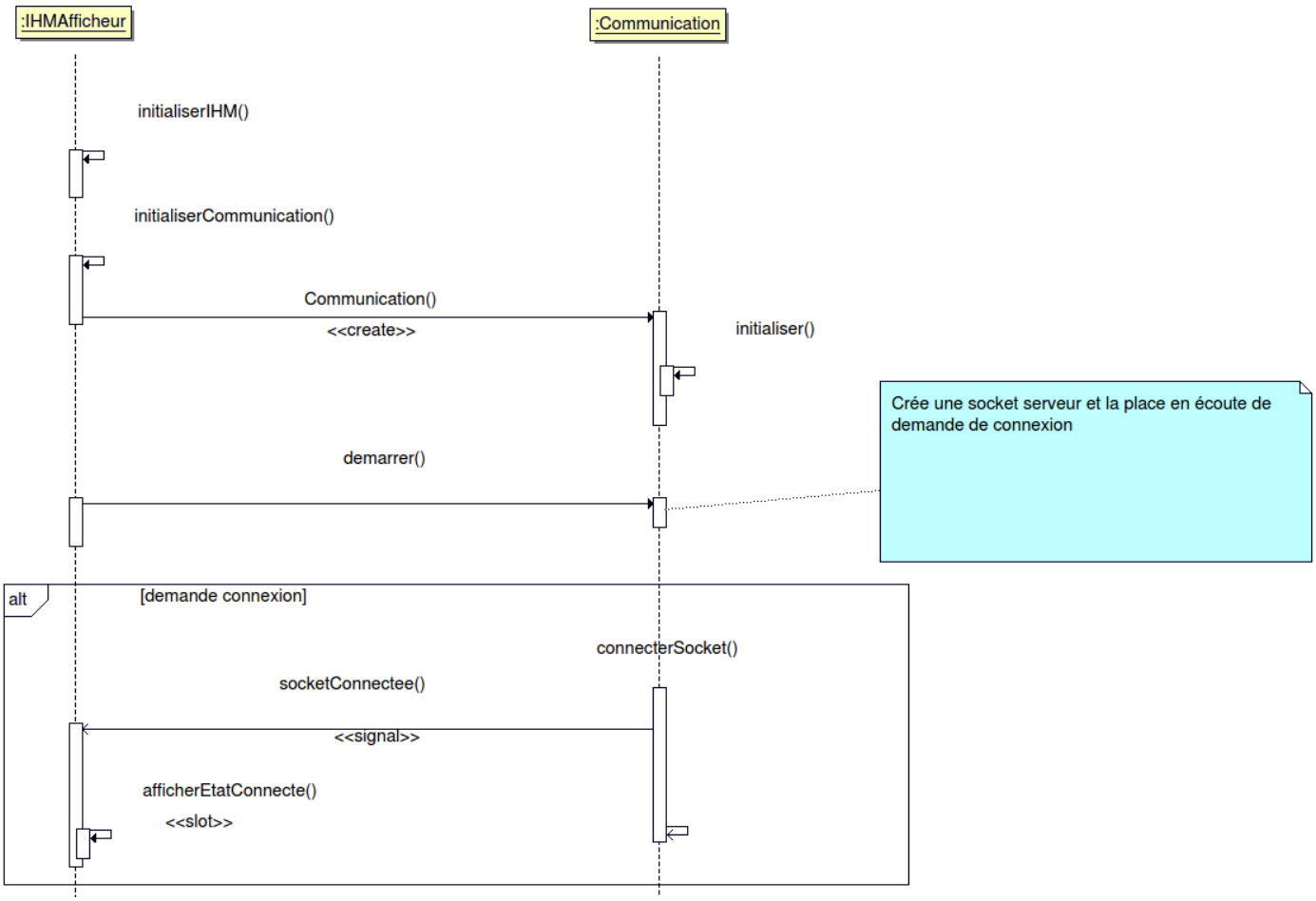
Les classes **Joueur** et **Club** ne servent qu'à stocker des informations avant de devoir les afficher telles que les noms des clubs et joueurs, les points et manches remportées par le joueur dans une partie etc...

La classe **Communication** est la classe assurant la réception et le traitement des trames, plusieurs choses sont à noter :
Toutes les méthodes en dessous de la méthode **lireTrame()** sont des **SIGNAUX** reliés pour la plupart à des **SLOTS** de la classe **IHM**.
Cette classe fait usage de classes proposées par Qt pour la connexion bluetooth telles que QBluetoothLocalDevice ou encore QBluetoothServer.
trames est un attribut de type QString contenant l'ensemble des trames reçues tandis que **trame** est un attribut contenant uniquement une seule trame afin de la traiter puis de s'en débarrasser pour passer à la suivante.

La classe **IHMAfficheur** est une classe qui, grâce à l'attribut "ui" généré par Qt Designer, est utilisée pour modifier l'affichage, c'est le premier objet créé au lancement de l'application qui initialisera par conséquent la communication.
Elle contient de nombreux **SLOTS** connectés aux **SIGNAUX** de la classe communication.

3.9. Diagrammes de séquence

Démarrage de l'application :



Dans ce diagramme de séquence la méthode initialiserIHM paramètre la fenêtre principale afin qu'elle soit noire et que toute les labels soient blanc afin d'améliorer la lisibilité et de rendre l'affichage plus agréable

```
void IHMAfficheur::initialiserIHM()
{
    ui->setupUi(this);

    ui->centralWidget->setStyleSheet("QWidget#centralWidget {background-color:
black;} QLabel {color: white}");

    // Mode debug
    QAction *quitter = new QAction(this);
    quitter->setShortcut(QKeySequence(QKeySequence(Qt::Key_Escape)));
    addAction(quitter);
    connect(quitter, SIGNAL(triggered()), this, SLOT(close()));
}
```

Les quatre dernières lignes permettent, lorsqu'un clavier est connecté d'assigner la touche "échap" a la sortie du programme pour faciliter le débogage de l'application.

Après l'instanciation de l'objet Communication sa méthode demarrer est instantanément appelée

```
void Communication::demarrer()
{
   // vérifier la présence du Bluetooth
   if (peripheriqueLocal.isValid() && serveur == nullptr)
   {
       // créer une socket serveur
       serveur = new QBluetoothServer(QBluetoothServiceInfo::RfcommProtocol, this);

       // connecter les signaux et les slots
       connect(serveur, SIGNAL(newConnection()), this, SLOT(connecterSocket()));

       // placer le serveur en écoute
       QBluetoothUuid uuid = QBluetoothUuid(uuidService);
       serviceInfo = serveur->listen(uuid, nomService);

       qDebug() << Q_FUNC_INFO << "Attente de connexion";
   }
   else
       qDebug() << Q_FUNC_INFO << "Bluetooth non disponible !";
}
```

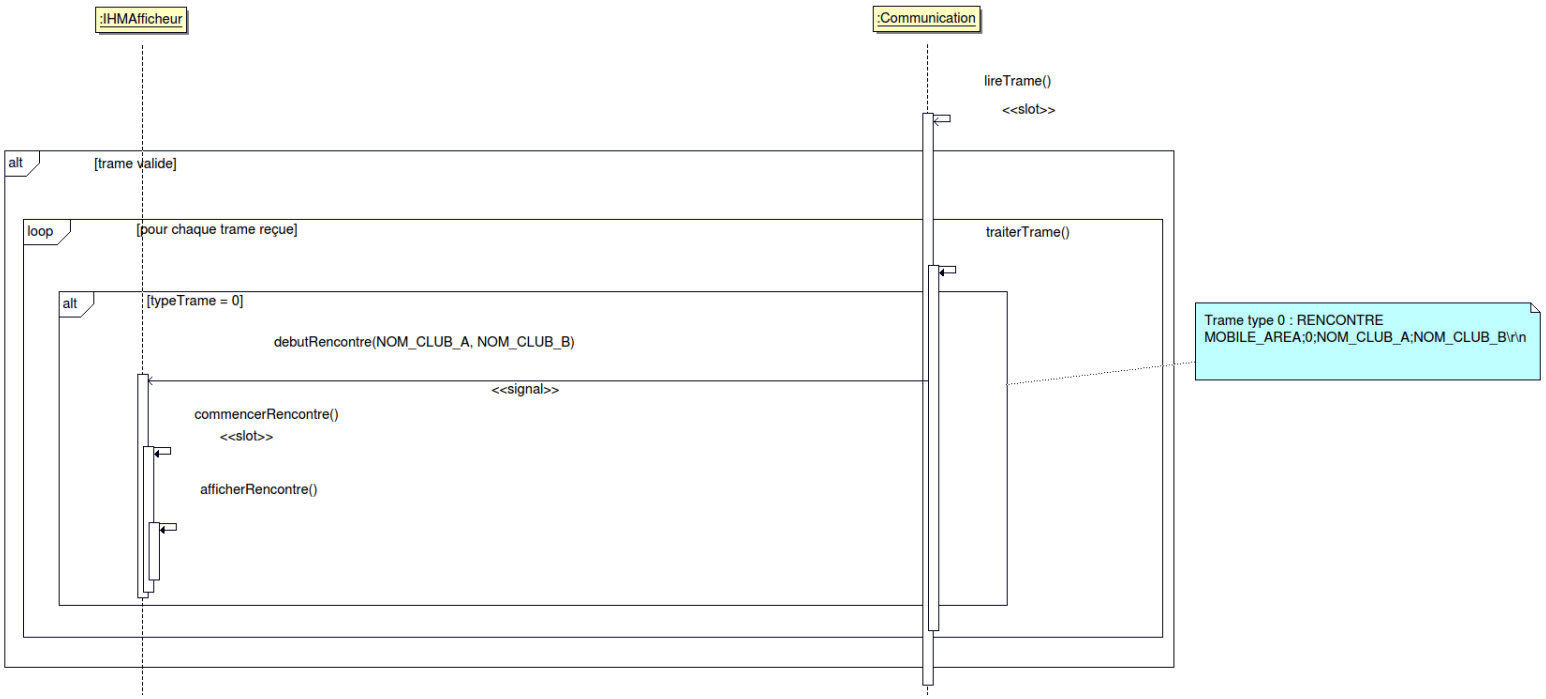
La ligne

```
QBluetoothUuid uuid = QBluetoothUuid(uuidService);
```

permet de récupérer l'identifiant universel unique (Universally unique identifier UUID)
qui sera utilisé afin de passer le socket serveur en mode écoute.

Un signal socketConnectee() est envoyé et l'utilisateur est notifié de la nouvelle connection entrante

Traitement d'une trame de création de rencontre :



Ce diagramme de séquence est au cœur de l'application, c'est l'exemple type de la réception de toutes les trames par le système.

tout d'abord le socket avertis qu'une nouvelle trame est disponible.

```
void Communication::lireTrame()
{
   QByteArray donnees;
   donnees = socket->readAll();
   trame += QString(donnees.data());

   if(trame.startsWith(ENTETE_TRAME) && trame.endsWith("\r\n"))
   {
       QStringList trames = trame.split("\r\n", QString::SkipEmptyParts);
       for(int i = 0; i < trames.count(); ++i)
       {
           qDebug() << Q_FUNC_INFO << i << trames[i];
           infosTrame = trames[i].split(";");
           traiterTrame(infosTrame);
       }
       trame.clear();
   }
}
```

Les données sont lues dans le socket par socket->readAll() et enregistrées dans un tableau d'octets (QByteArray), ce tableau est cast en chaîne de caractères (QString) afin de pouvoir être traitée plus facilement.

Le programme vérifie d'abord que la trame lui parvient bien du bon module en vérifiant que la trame débute bien par "MOBILE_AREA" :

```
trame.startsWith(ENTETE_TRAME)
```

Puis il vérifie qu'elle est bien complète, c'est à dire qu'elle se termine par "\r\n" (norme commune pour la communication au sein du projet):

```
trame.endsWith("\r\n")
```

La trame est ensuite séparée au niveau des ";" puis affectés dans une liste de chaînes de caractères (QStringList)
cette liste est passée en parametres de la méthode

```
 traiterTrame(infosTrame);
```

```
bool Communication::traiterTrame(QStringList infosTrame)
{
   if(infosTrame.count() < CHAMP_TYPE_TRAME)
       return false;

   qDebug() << Q_FUNC_INFO << infosTrame;

   switch(infosTrame[CHAMP_TYPE_TRAME].toInt())
   {
       case TypeTrame::RENCONTRE:
           emit(debutRencontre(infosTrame[NOM_CLUB_1], infosTrame[NOM_CLUB_2]));
           break;
       ...
    }
```

Dans ce cas scénario la trame reçue est de type :

MOBILE_AREA;**RENCONTRE**;NOM_CLUB_A;NOM_CLUB_B\r\n

Elle correspond donc au "case" RENCONTRE

Le signal de début de rencontre est déclenché et fait donc appel au slot commencerRencontre de la classe IHMAfficheur.

```
void IHMAfficheur::commencerRencontre(QString club1, QString club2)
{
   rencontre = new Rencontre(club1, club2);
   afficherRencontre();

   connect(communication, SIGNAL(creationPartieSimple(QStringList)), rencontre,
SLOT(creerPartieSimple(QStringList)));
   connect(communication, SIGNAL(creationPartieDouble(QStringList)), rencontre,
SLOT(creerPartieDouble(QStringList)));
   connect(communication, SIGNAL(nouveauScorePartie(int, int, int, int, int)), this,
SLOT(actualiserScore(int, int, int, int, int)));
   connect(communication, SIGNAL(changementEtatPartie(int,QString)), this,
SLOT(changerEtatPartie(int,QString)));
   connect(communication, SIGNAL(detectionNET(int)), this, SLOT(afficherNET(int)));
}
```

La rencontre est créée, les clubs sont renseignés, et l'affichage est appelé avant de réaliser toute les connexions afin d'assurer le fonctionnement de l'application.

Ceci signifie donc que la plupart des fonctionnalités ne sont pas disponibles tant qu'aucune rencontre n'est créée, ceci agit comme une précaution.

```
void IHMAfficheur::afficherRencontre()
{
   qDebug() << Q_FUNC_INFO;
   ui->nomEquipe1->setText(rencontre->getNomEquipe1());
   ui->nomEquipe1->setAlignment(Qt::AlignCenter);
   ui->nomEquipe2->setText(rencontre->getNomEquipe2());
   ui->nomEquipe2->setAlignment(Qt::AlignCenter);
}
```

La méthode afficherRencontre ne fait qu'actualiser les noms des clubs sur l'affichage dans leurs labels correspondants "nomEquipe1" et nomEquipe2"

3.10. Le protocole de communication

L'afficheur ne reçoit que des trames provenant d'un (ou plusieurs) module(s) mobile(s).

3.10.1. Format des trames

L'afficheur recevra différents types de trames avec chacune un format prédéfini.
Pour la création d'une partie :

MOBILE_AREA;RENCONTRE;NOM_CLUB_A;NOM_CLUB_B\r\n

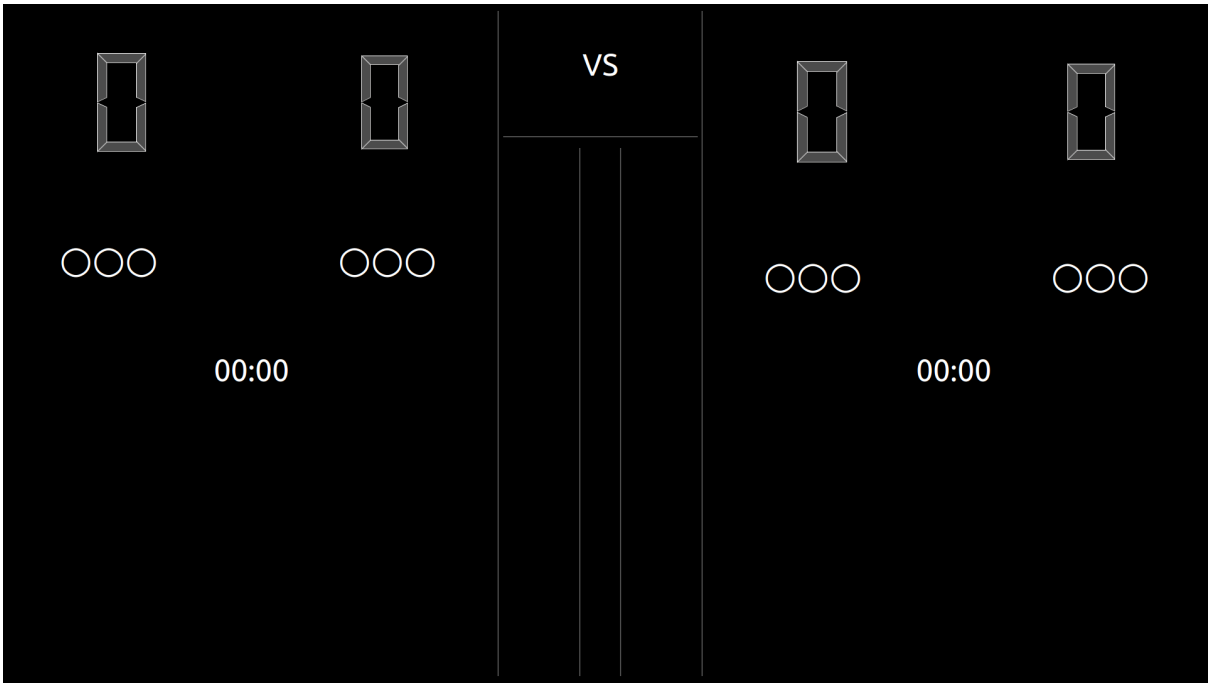
Pour la gestion du score d'une partie :

MOBILE_AREA;INFO_PARTIE;ID_PARTIE;NOM_JOUEUR_A;PRENOM_JOUEUR_A;[NOM_DEUXIEME_JOUEUR_A];[PRENOM_DEUXIEME_JOUEUR_A];NOM_JOUEUR_B;PRENOM_JOUEUR_B;[NOM_DEUXIEME_JOUEUR_B];[PRENOM_DEUXIEME_JOUEUR_B]\r\n

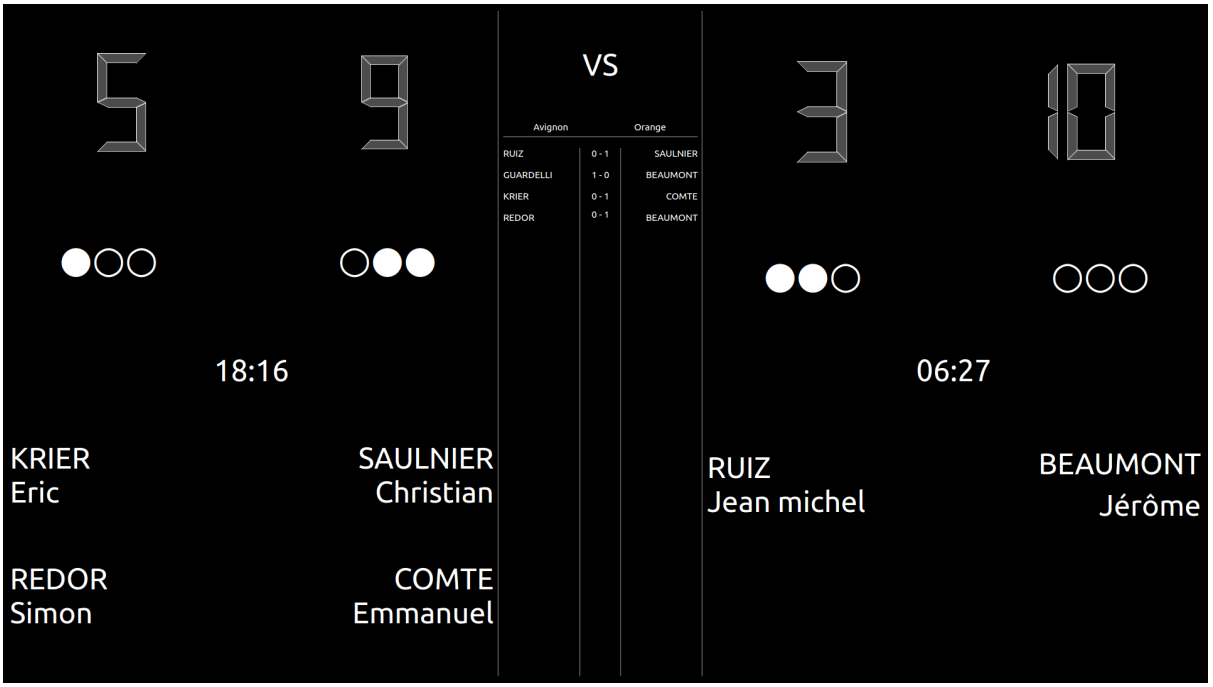
Pour démarrer / Terminer une partie :

MOBILE_AREA;3;ID_PARTIE;ETAT\r\n

3.11 Maquette de l'IHM



Ceci est l'IHM vide de l'application, a l'allumage de la raspberry pi si aucun module mobile AREA n'est disponible l'affichage est instancié ainsi.



Et ceci est un exemple de l'IHM en plein fonctionnement.
Les deux informations mises en évidence sont les noms des joueurs participant à une partie, ainsi que le score de la partie.
Les manches marquées par les joueurs de chaque club sont désignées par des points blancs.

Au centre se trouve l'historique des parties déjà terminées, chaque fois qu'un terminal mobile envoie une trame de fin de partie, la partie correspondant à l'identifiant reçu est terminée: les noms des joueurs s'ajoutent à l'historique et le score est ajouté (1 point pour le vainqueur 0 pour le perdant).

3.12 Tests de validation

Désignation	Résultat attendu	Validation
Afficher le score en temps réel	Les scores s'actualise avec le mobile AREA	✓
Afficher les manches en temps réel	Les manches s'actualisent en même temps que celles du mobile AREA	✓
Afficher les participants de chaque parties	Les noms des participants s'affichent ou se cachent selon si la partie est en cours ou terminée	✓
Gérer l'affichage sur plusieurs parties simultanément	Les deux parties s'actualisent indépendamment et sont entièrement fonctionnelles	✓
Afficher les informations sur la rencontre	les noms des clubs s'affrontent sont affichés en tout temps	✓
Afficher l'historique des parties	Chaque partie terminée s'ajoute dans l'historique des parties	✓
La gestion du temps est entièrement fonctionnelle	Le chronomètre s'active au début d'une partie et s'arrête à la fin	X
Les temps morts sont fonctionnels	Un temps mort peut être appelé et la partie reprend a la fin du temps mort	X

4. Présentation personnelle : BRUNET Bastien

4.1 Objectifs

Le module Mobile-AREA est une application pour terminal mobile (Tablette Android) qui permet à l'arbitre de gérer la partie en cours (score, temps morts, ...) ainsi que le déroulement d'une rencontre.

4.2 Mise en situation

La Fédération Française de Tennis de Table (FFTT) fournit des feuilles de rencontre pour organiser et gérer les compétitions par équipes, l'application est conçue pour rester fidèle aux feuilles de rencontres papier tout en simplifiant les tâches répétitives.

N° 03220128	association : UP Plédranaise 1			
N° Licence	NOM - PRENOM	Points	Mute Etranger	Cartons
225997	A CUREC Jerome	1089		
2212888	B BLANC Patrice	879		
228842	C PEDRON Michel	820		
2214476	D ROUXEL Nicolas	994		

Section dédiée aux joueurs dans une feuille de rencontre

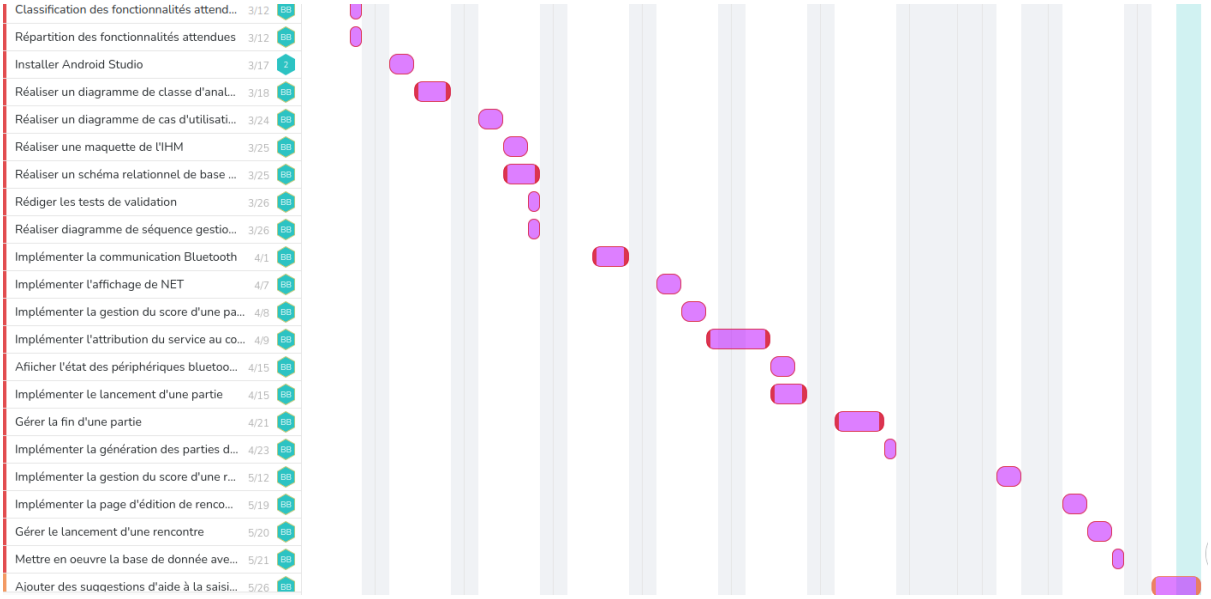
On peut voir ici les informations nécessaires à l'inscription d'une équipe, pour l'application on gardera simplement le numéro de licence, le nom et le prénom de chaque joueur ainsi que le nom du club que l'équipe représente. On peut aussi relever qu'une lettre est associée à chaque joueur, cette dernière est utilisée pour l'organisation des parties .

SCORES					ORDRE DES PARTIES					POINTS	
1	2	3	4	5						ABCD	WXYZ
06	09	08			A	CUREC Jerome	contre	W	DIVEU Erik	2	1
01	-07	02	06		B	BLANC Patrice	"	X	BAUDOUARD Daniel	2	1
05	10	-09	-10	-07	C	PEDRON Michel	"	Y	PHILIPPE Jean Yves	1	2
02	04	05			D	ROUXEL Nicolas	"	Z	BERVET Christophe	2	1
-08	01	03	07		A	CUREC Jerome	"	X	BAUDOUARD Daniel	2	1
-07	06	03	03		B	BLANC Patrice	"	W	DIVEU Erik	2	1
06	08	01			D	ROUXEL Nicolas	"	Y	PHILIPPE Jean Yves	2	1
06	07	06			C	PEDRON Michel	"	Z	BERVET Christophe	2	1
05	-08	-05	-06		AB	CUREC Jerome / BLANC Patrice	"	WY	DIVEU Erik / PHILIPPE Jean Yves	1	2
06	02	01			CD	PEDRON Michel / ROUXEL Nicolas	"	XZ	BAUDOUARD Daniel / BERVET Christophe	2	1
-09	-06	-09			D	ROUXEL Nicolas	"	W	DIVEU Erik	1	2
03	05	-03	04		C	PEDRON Michel	"	X	BAUDOUARD Daniel	2	1
06	03	06			A	CUREC Jerome	"	Z	BERVET Christophe	2	1
07	05	06			B	BLANC Patrice	"	Y	PHILIPPE Jean Yves	2	1
07	09	12			C	PEDRON Michel	"	W	DIVEU Erik	2	1
03	05	05			D	ROUXEL Nicolas	"	X	BAUDOUARD Daniel	2	1
08	-05	07	07		A	CUREC Jerome	"	Y	PHILIPPE Jean Yves	2	1
06	07	-10	08		B	BLANC Patrice	"	Z	BERVET Christophe	2	1
									TOTAL DES POINTS DE CHAQUE EQUIPE	33	21

Section dédiée à l'organisation des parties

L'organisation des parties dans la rencontre se fait grâce aux lettres associées à chacun des joueurs, l'une des deux équipes est associée aux lettres ABCD et l'autre aux lettres WXYZ. Les parties se déroulent dans l'ordre donné dans la grille. On voit qu'une victoire de partie rapporte un certain nombre de points. L'équipe ayant accumulé le plus de points remporte la rencontre.

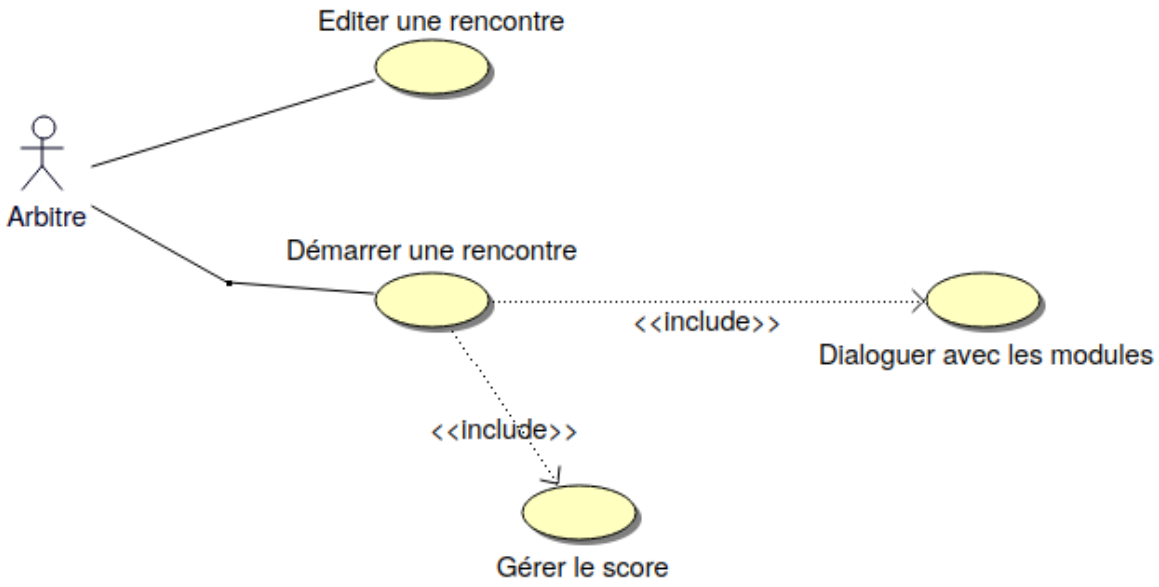
4.3 Planification



4.4 Répartition des tâches

Itération	Priorité	Tâche à réaliser
2	Haute	Démarrer et gérer le score d'une partie
2	Haute	Communiquer avec le module Net_AREA
2	Haute	Communiquer avec le module Afficheur_AREA
2	Haute	Afficher une détection de net
2	Moyenne	Démarrer une rencontre
2	Moyenne	Editer les informations d'une rencontre
2	Moyenne	Sauvegarder les information d'une rencontre dans une base de données
2	Moyenne	Afficher qui est le serveur et qui est le relanceur
3	Basse	Gérer les temps morts d'une partie
3	Basse	Afficher les connexions aux modules

4.5 Diagramme des cas d'utilisation



L'arbitre doit pouvoir éditer les informations d'une rencontre (nom des joueurs, numéro de licence, nom du club et le nombre de manches gagnantes pour cette rencontre). Ces informations pourront être sauvegardées dans une base de données afin de les réutiliser. Ensuite, il démarre une rencontre. Ce module communique en Bluetooth avec les deux autres modules.

Pendant une rencontre, l'arbitre gère le score en validant les points pour chaque échange. Il peut être "aidé" pour une détection de "net" lors d'un service.

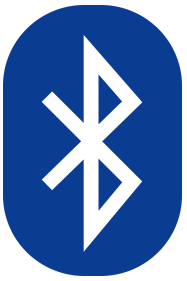
4.6 Outils de développement

Désignation	*Caractéristiques*
OS Poste de développement	PC sous Windows © ou GNU/Linux Ubuntu
EDI et langage	Android Studio (Java)
SGBDR	SQLite3
OS Tablette	Android ©

Caractéristiques techniques de la tablette tactile :

Modèle	Processeur Graphique	Mémoire vive (RAM)	Batterie	Système d'exploitation	Bluetooth
Galaxy Tab S2 (SM-T813)	Qualcomm Adreno 510,550 MHz	3 Go	5870 mAh	Android 7.0	4.1

4.7 Présentation du Bluetooth



Bluetooth est une norme de communications permettant l'échange bidirectionnel de données à très courte distance en utilisant des ondes radio UHF sur une bande de fréquence ISM (Industrial, Scientific and Medical) 2,4 GHz.

Il opère dans la bande de fréquences comprise entre 2 400 et 2 483,5 MHz. Les 79 canaux RF sont numérotés de 0 à 78 et séparés de 1 MHz en commençant par 2 402 MHz.

Le système Bluetooth utilise une modulation de fréquence(FSK) avec une rapidité de modulation de 1 Mbaud.

Il fonctionne sur un modèle maître/esclave ou périphérique maître peut administrer 7 esclaves actifs.

Il existe trois classes de modules radio Bluetooth sur le marché :

Classe	Puissance	Portée
1	100 mW (20 dBm)	100 mètres
2	2,5 mW (4 dBm)	10 à 20 mètres
3	1 mW (0 dBm)	Quelques mètres

La plupart des fabricants d'appareils électroniques dont notre tablette utilisent des modules classe 2.

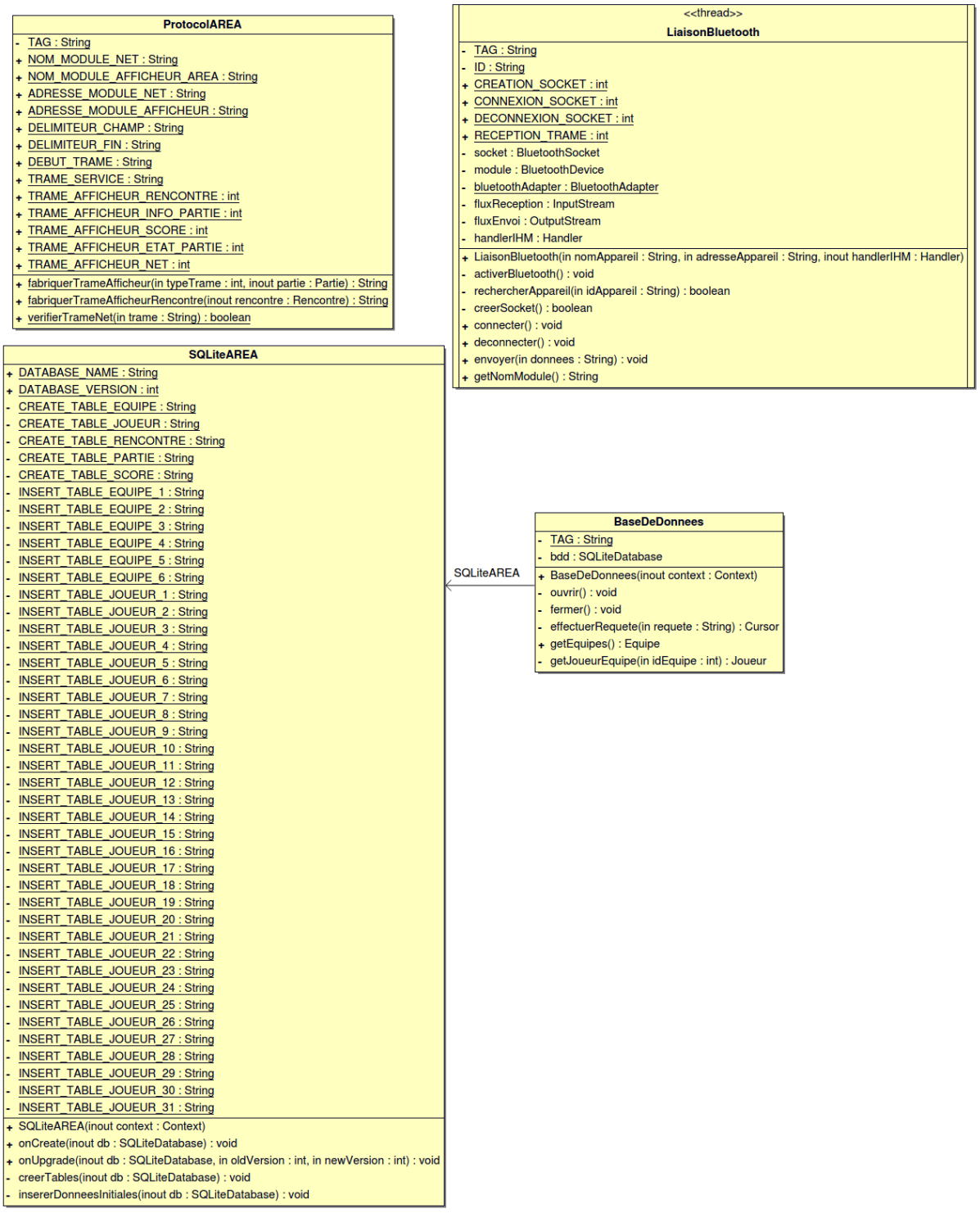
4.8 Diagramme de classes

La classe **Equipe** permet de modéliser une équipe de plusieurs joueurs et de gérer son score au cours d'une rencontre.

La classe **Partie** permet la gestion du score d'une partie.

La classe **Rencontre** s'occupe de l'organisation des parties d'une rencontre.

La classe **Joueur** modélise un joueur grâce à son nom, son prénom et un numéro de licence.

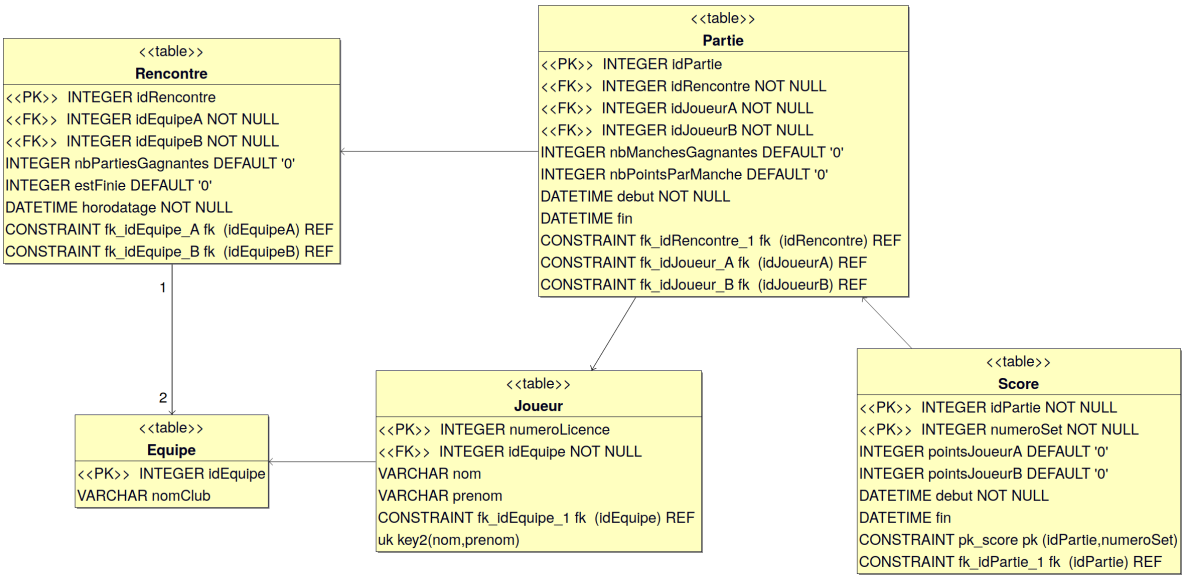


La classe **LiaisonBluetooth** permet la communication avec les modules net et afficheur.

La classe **ProtocolAREA** regroupe les informations et méthodes relatives au protocole de communication.

Les classes **SQLiteAREA** et **BaseDeDonnees** d'interagir avec la base de données.

4.9 Spécification de la base de données



La table **Rencontre** assure le stockage des informations d'une rencontre, les deux clés étrangères idEquipeA et idEquipeB font référence aux deux équipes qui s'affrontent à l'occasion de cette rencontre.

La table **Equipe** se caractérise par un nom de club et une clé primaire idEquipe.

La table **Joueur** permet de représenter un joueur qui possède comme clé primaire un numéro de licence et un nom et un prénom. Une clé étrangère sur la table équipe permet de savoir à quelle équipe le joueur est associé.

La table **Partie** permet l'enregistrement d'une partie dans la base de données d'une partie horodatée

La table **Score** assure l'enregistrement des scores des différentes manches d'une partie

Exemples de requêtes SQL :

Liste des clubs :

```
SELECT * FROM Equipe;
```

Cette requête SQL utilise un simple SELECT afin de sélectionner tous les enregistrements (*) de la table Equipe, il n'y a pas de conditions .
Liste des joueurs pour un club :

```
SELECT * FROM Joueur INNER JOIN Equipe ON
Joueur.idEquipe=Equipe.idEquipe WHERE Equipe.idEquipe='3';
```

Cette requête est un peu plus complexe que la première, en effet il s'agit toujours d'un SELECT mais qui à la particularité d'être couplé avec un INNER JOIN qui permet d'effectuer une jointure intérieure entre deux tables, ici il s'agit des tables Equipe et Joueur.

La jointure se fait sur la clé primaire de la table Equipe 'idEquipe' et la clé étrangère du même nom de la table Joueur.

La clause WHERE permet d'ajouter une condition à la requête, dans cet exemple on sélectionne tous les joueurs de l'équipe ayant pour identifiant 3.

4.10 Protocole de communication

Le module Mobile_AREA doit communiquer en Bluetooth avec les deux autres modules qui composent le système.

4.10.1 Format des trames de communication avec le module Net_AREA

Deux scénarios sont possibles :

- L'initialisation d'une séquence de NET (envoi de données):

  MOBILE_AREA;SERVICE\r\n

- La détection d'une séquence de NET (réception de données):

  NET_AREA;NET\r\n

4.10.2 Format des trames de communication avec le module Afficheur_AREA

Envoi des informations d'une partie :

MOBILE_AREA;1;ID_PARTIE;NOM_JOUEUR_A;PRENOM_JOUEUR_A;[NOM_DEUXIEME_JOUEUR_A];[PRENOM_DEUXIEME_JOUEUR_A];NOM_JOUEUR_B;PRENOM_JOUEUR_B;[NOM_DEUXIEME_JOUEUR_B];[PRENOM_DEUXIEME_JOUEUR_B]\r\n

Gestion du score d'une partie :

MOBILE_AREA; ID_PARTIE;POINTS_JOUEUR_A;POINTS_JOUEUR_B; NB_MANCHES_GAGNEES_JOUEUR_A;NB_MANCHES_GAGNEES_JOUEUR_B\r\n

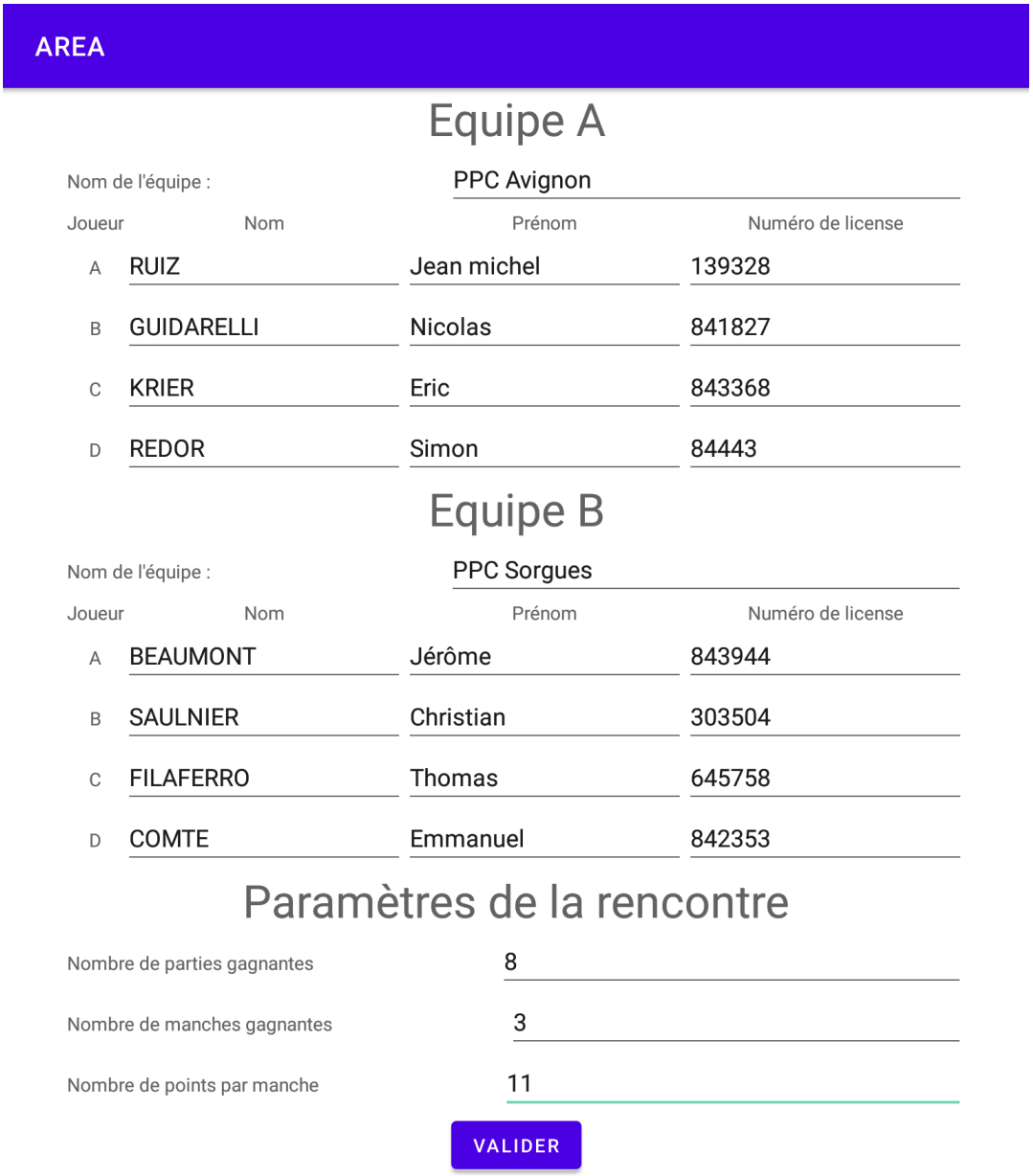
Démarrer / Terminer une partie :

MOBILE_AREA; ID_PARTIE;ETAT\r\n

Détection d'un NET :

MOBILE_AREA;4;ID_PARTIE\r\n

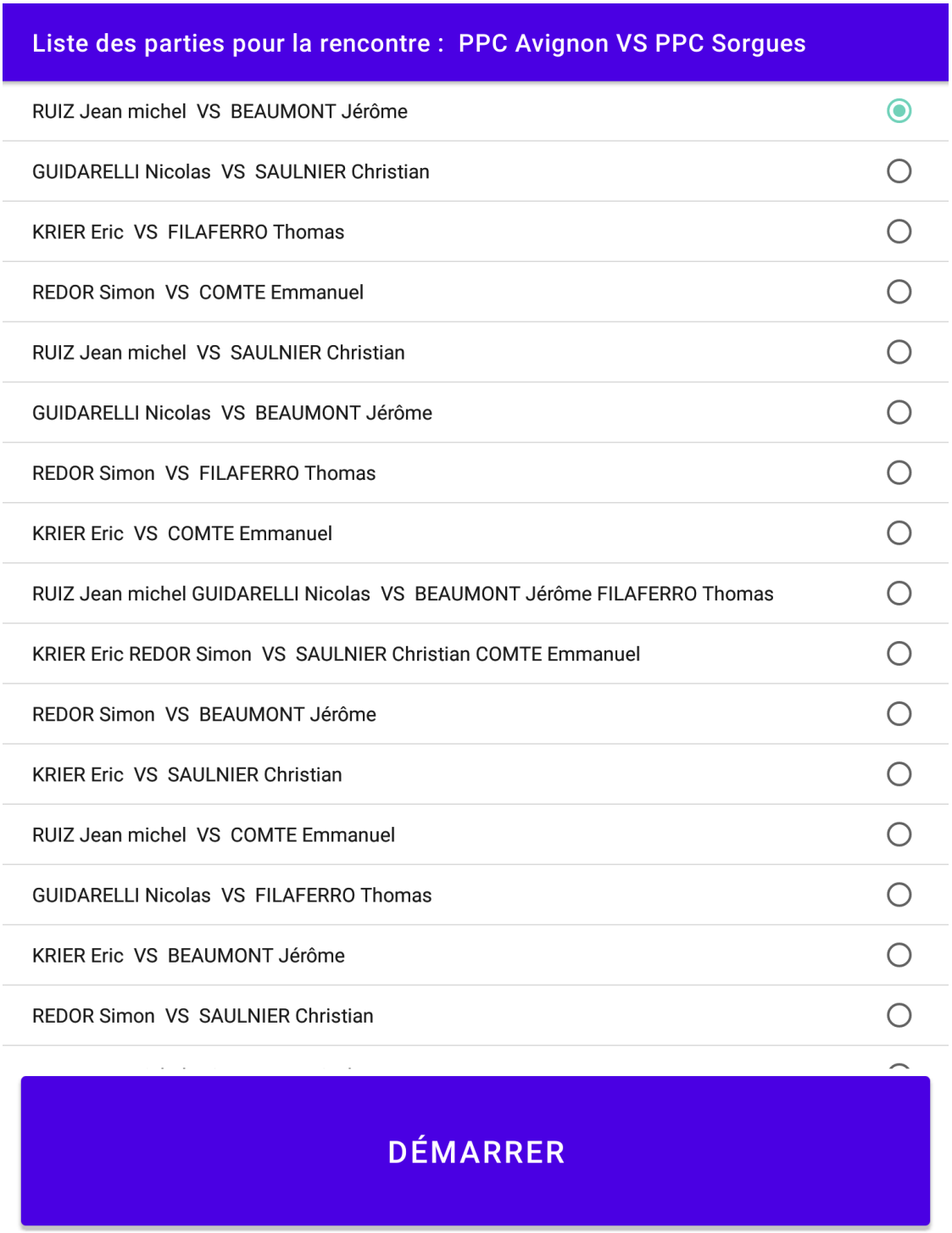
4.11 Présentation de l'IHM



L'écran d'édition de rencontre permet la saisie des informations relatives à une rencontre ainsi que le paramétrage de la rencontre et de ses parties, il s'agit de la page qui s'affiche au lancement de l'application. L'écran se divise en trois sections :

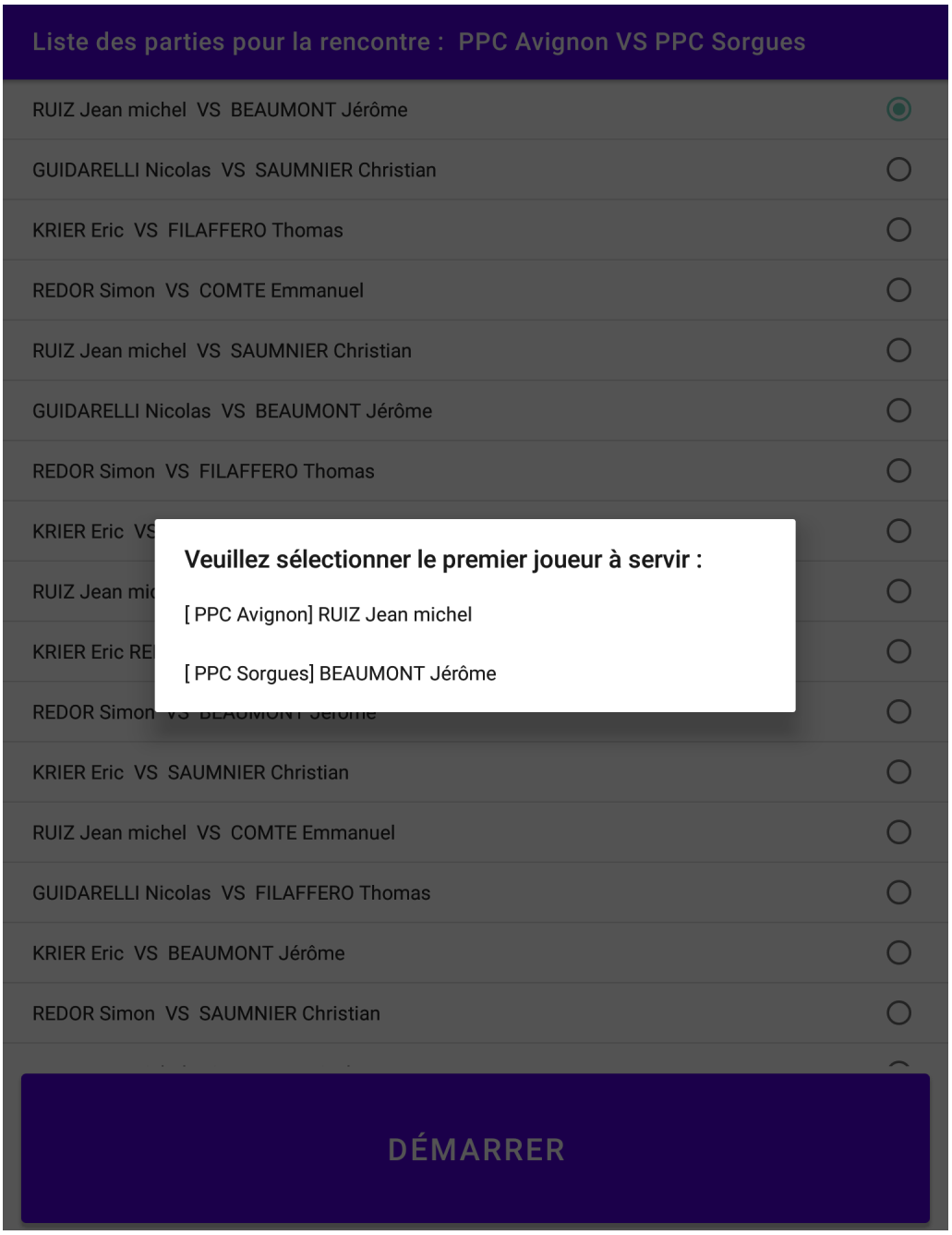
- Deux sections dédiés à l'édition des informations des équipes, à savoir :
  - Le nom de l'équipe
  - Pour chacun des 4 joueurs de l'équipe :
    - Nom
    - Prénom
    - Numéro de licence

- Le paramétrage de la rencontre et de ses parties :
  - Le nombre de parties gagnantes
  - Le nombre de manches gagnantes
  - Le nombre de points par manche



Les parties sont générées après la validation des informations de la rencontre saisies par l'arbitre, elles sont toutes affichées dans cette page de l'application qui permet de choisir et démarrer une partie. On remarque la présence de parties doubles pour correspondre au maximum avec une feuille de rencontre.

Lorsqu'une partie est terminée elle s'affiche en grisée et il n'est plus possible de la sélectionner pour la lancer.



Lorsque l'arbitre sélectionne une partie et appuie sur le bouton 'Démarrer', une boîte de dialogue lui demande de sélectionner le joueur qui va servir en premier, entre crochets se trouve le nom de l'équipe du joueur puis son nom suivi de son prénom.

Ci-dessus l'interface permettant de gérer le score d'une partie.

Chaque côté permet de gérer le score d'un joueur grâce aux boutons '+1' et '-1' qui permettent respectivement d'ajouter ou de retirer un point au joueur.

La visualisation du score se fait au centre de l'écran, les chiffres séparés par un tiret permettent de visualiser le score de la manche en cours et les chiffres les plus en bas de visualiser le nombre de manches gagnées par chacun des joueurs.

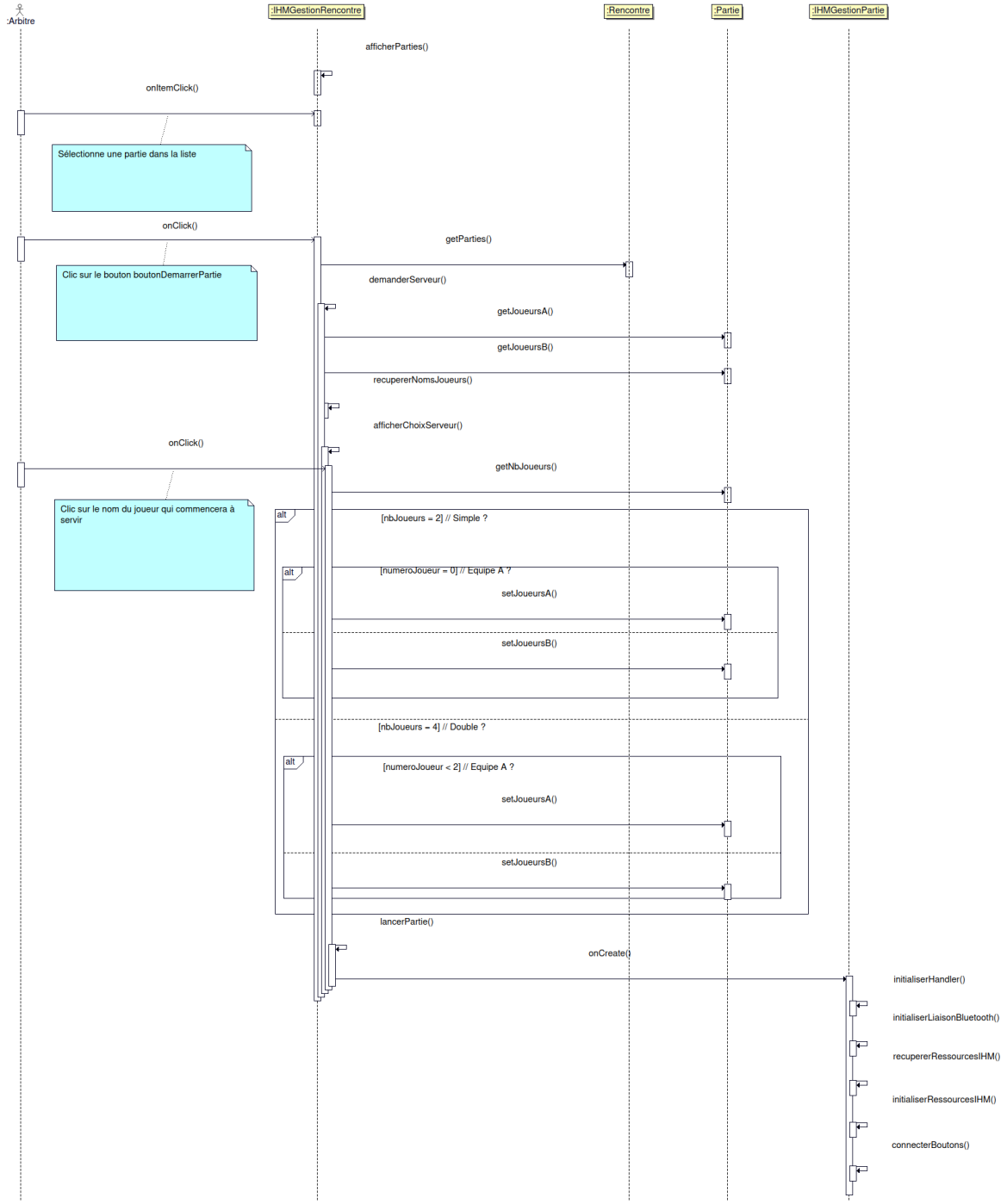
Dans la partie centrale haute de l'écran se trouve le message NET indiquant qu'un NET a été détecté par le module dédié, il s'affiche puis disparaît après un court instant .

L'identification et la désignation du serveur est géré automatiquement par l'application à côté du nom du joueur qui doit servir se trouve une icône.

En bas au centre de l'écran se trouve l'affichage des connexions aux autres modules du projet, une pastille verte signifie que le module est connecté et une pastille rouge que le module est déconnecté.

4.12 Scénarios

4.12.1 Lancer une partie



On commence par afficher les parties à l'aide de la méthode afficherParties() de la classe IHMGestionRencontre.

Puis une fois que l'utilisateur a sélectionné une partie et qu'il a appuyé sur le bouton "DémarrerPartie" on récupère les parties afin de déterminer laquelle doit être lancée.

Ensuite on appelle la méthode demanderServeur() qui va récupérer les joueurs de la partie en question extraire leur noms avec la méthode recupererNomsJoueur() puis la méthode afficherChoixServeur() affichera une boîte de dialogue permettant à l'utilisateur de choisir quel joueur sera le premier à servir.

Une fois que le serveur est sélectionné on cherche à déterminer si la partie est double ou simple et ensuite dans quel équipe il se trouve, une fois fait on le définit comme étant serveur et on le remplace dans la partie.

Ensuite on peut lancer la partie avec la méthode lancerPartie() de la classe IHMGestionRencontre qui va lancer l'activité de gestion de partie en déclenchant par la même occasion la méthode onCreate() de la classe IHMGestionPartie qui va initialiser le handler du Bluetooth avec la méthode initialiserHandler() puis initialiser la liaison Bluetooth avec la méthode initialiserLiaisonBluetooth() et enfin connecter les boutons puis récupérer et initialiser les ressources de l'IHM grâce aux méthodes connecterBoutons(), recupererRessourcesIHM() et initialiserRessourceIHM().

Définition de la méthode lancerPartie() :

```
private void lancerPartie(Partie partie)
{
      final Intent intent = new Intent(IHMGestionRencontre.this,
IHMGestionPartie.class);
      intent.putExtra(ID_INTENT_LANCEMENT_PARTIE, partie);
      Log.d(TAG,"Lancement de l'activité IHMGestionPartie");
      startActivityForResult(intent, DEMARRAGE_PARTIE);
}
```

Dans cette méthode on initialise d'abord un objet intent de la classe Intent, cet objet va permettre à cette activité (IHMGestionRencontre) d'envoyer un message à une autre activité, ici IHMGestionPartie. Un objet partie de la classe Partie est ajouté à l'intent grâce à la méthode putEtxra(), Le lancemet de l'activité se fait grâce à la méthode startActivityForResult() qui prend en paramètre l'intent précédemment constitué.

Définition de la méthode onCreate():

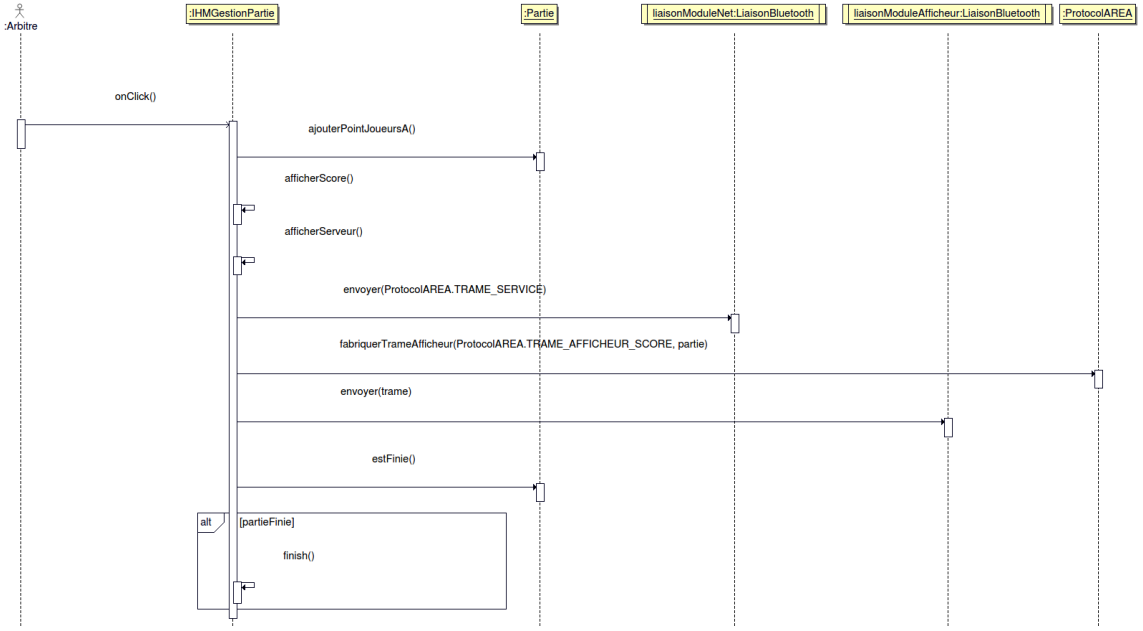
```
protected void onCreate(Bundle savedInstanceState)
{
	super.onCreate(savedInstanceState);
	setContentView(R.layout.ihm_gestion_partie);
	Log.d(TAG, "onCreate()");

	partie = (Partie)
getIntent().getSerializableExtra(IHMGestionRencontre.ID_INTENT_LANCEMENT
_PARTIE);

	initialiserHandler();

	initialiserLiaisonBluetooth();

	recupererRessourcesIHM();

	initialiserRessourcesIHM();

	connecterBoutons();
}
```

La première partie du code permet de sauvegarder l'état de l'activité lorsque celle ci redémarre, il faut savoir que sous Android lorsque l'on change l'orientation de la tablette, les applications en cours d'exécution redémarrent ce qui explique la présence d'un paramètre savedInstanceState de type Bundle et de l'appel du constructeur de la classe parent auquel on passe ce paramètre qui permettent de sauvegarder l'état de l'application. La ligne suivante initialise l'attribut partie de la classe IHMGestionPartie avec la partie ajouté à l'intent envoyé par l'activité IHMGestionRencontre, pour le récupérer, les méthodes getIntent() et getSerializableExtra() sont appelées, l'identifiant de l'intent à récupérer est passé en paramètre de cette dernière méthode.

Voici le diagramme de classe du scénario :

4.12.2 Gérer le score

Pour l'ajout d'un point :



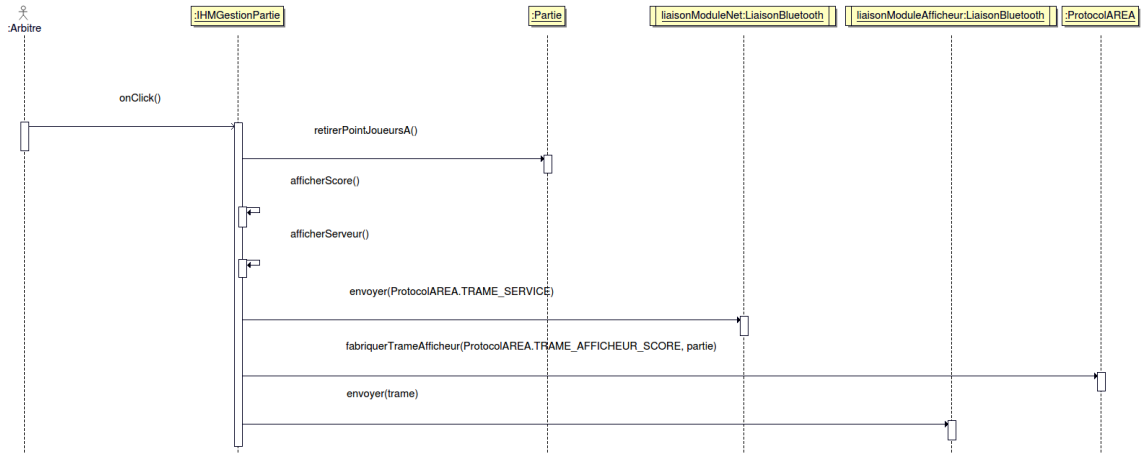
Lorsque l'arbitre appuie sur le bouton permettant d'ajouter un point (ici à joueursA) la méthode ajouterPointJoueursA() de la classe Partie est appelée afin d'incrémenter le score de joueursA.

Ensuite le score et la personne qui doit servir sont actualisés sur l'IHM grâce aux méthodes afficherScore() et afficherServeur() de la classe IHMGestionPartie.

Puis une trame permettant le passage en mode détection est fabriquée puis envoyée au module NET à l'aide des méthodes fabriquerTrameAfficheur() de la classe ProtocolAREA et envoyer() de la classe LiaisonBluetooth.

Enfin une trame de mise à jour du score est envoyé au module Afficheur puis si la partie est finie la méthode finish() permettant de terminer l'activité est appelée.

Pour retirer un point :



Lorsque l'arbitre appuie sur le bouton permettant de retirer un point (ici à joueursA) la méthode retirerPointJoueursA() de la classe Partie est appelée afin de décrémenter le score de joueursA.

Ensuite le score et la personne qui doit servir sont actualisés sur l'IHM grâce aux méthodes afficherScore() et afficherServeur() de la classe IHMGestionPartie.

Puis une trame permettant le passage en mode détection est fabriquée puis envoyée au module NET.

Enfin on fabrique une trame pour informer le module afficheur du changement de score avec la méthode fabriquerTrameAfficheur() de la classe ProtocolAREA puis on envoie cette trame au module afficheur en utilisant la méthode envoyer() de la classe LiaisonBluetooth.

Définition de la méthode envoyer() :

```
public void envoyer(String donnees)
{
   if (module == null || socket == null)
       return;

   new Thread()
   {
```

```
        @Override public void run()
        {
            try
            {
                if(socket.isConnected())
                {
                    Log.d(TAG,"Envoi vers le module " + module.getName()
+ " | Adresse : " + module.getAddress() + " | Données : " + donnees);
                    fluxEnvoi.write(donnees.getBytes());
                }
            }
            catch(IOException e)
            {
                Log.e(TAG,"Erreur envoi socket");
                e.printStackTrace();
            }
        }
    }.start();
}
```

Cette méthode de la classe LiaisonBluetooth prend en paramètre une chaîne de caractère représentant les données à envoyer.

```
//Attributs de la classe LiaisonBluetooth
private BluetoothSocket socket = null;
private BluetoothDevice module = null;
```

On commence par vérifier que les attributs module et socket sont bien initialisés, si ils ne le sont pas on sort de la méthode. Ces attributs sont essentiels au bon fonctionnement de la méthode puisque l'objet socket de type BluetoothSocket représente la connexion et que l'objet module de la classe BluetoothDevice représente le module auquel le socket est connecté.

Ensuite on crée un nouveau Thread puis on redéfinit la méthode run() qui sera celle appelée au démarrage du Thread.

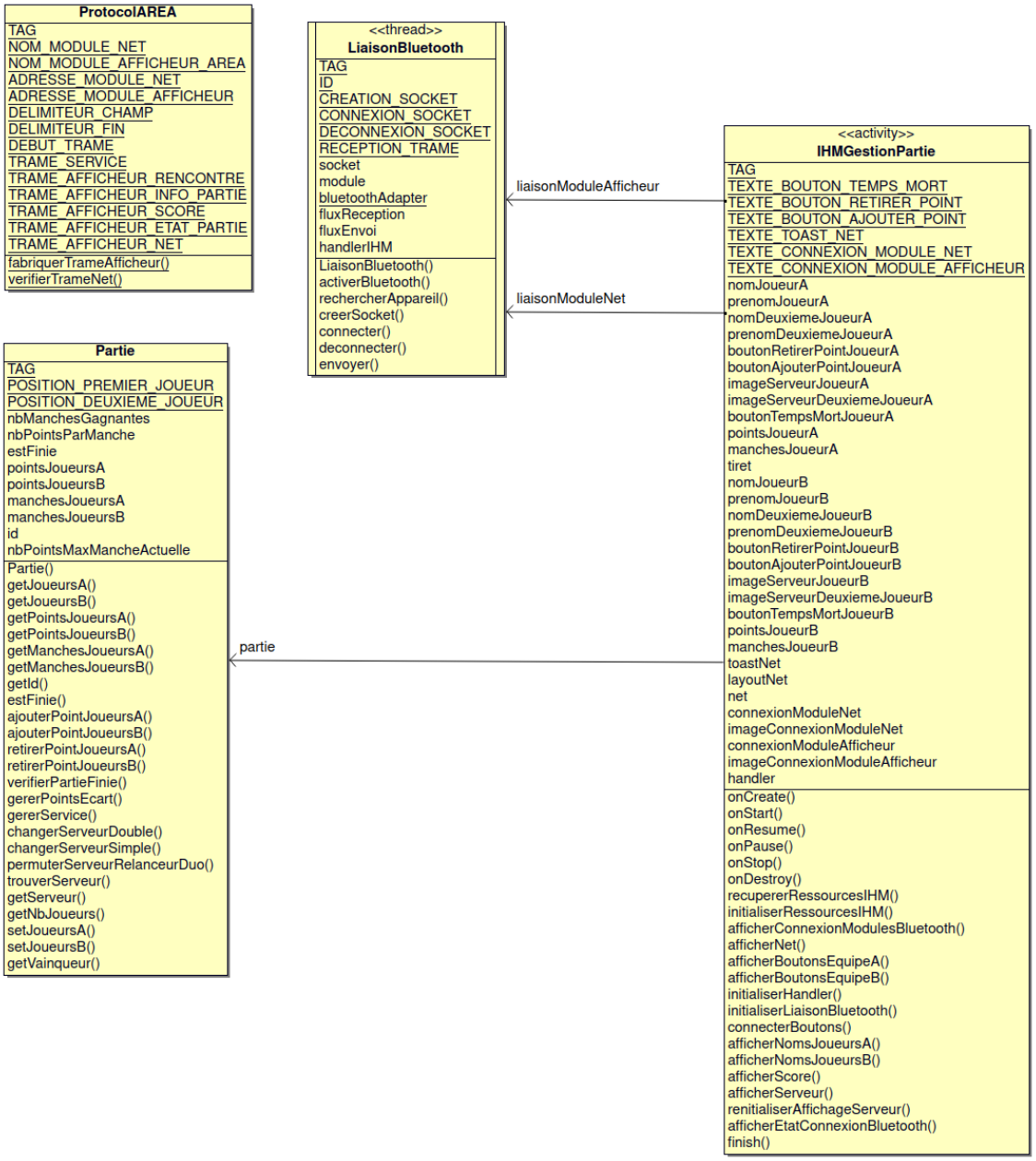
```
//Attributs de la classe LiaisonBluetooth
private InputStream fluxReception = null;
private OutputStream fluxEnvoi = null;
```

Pour communiquer en Bluetooth la classe LiaisonBluetooth utilise un flux d'entrée de type InputStream, utilisé pour la réception de donnée, et un flux de sortie de type OutputStream, utilisé pour l'envoi de données.

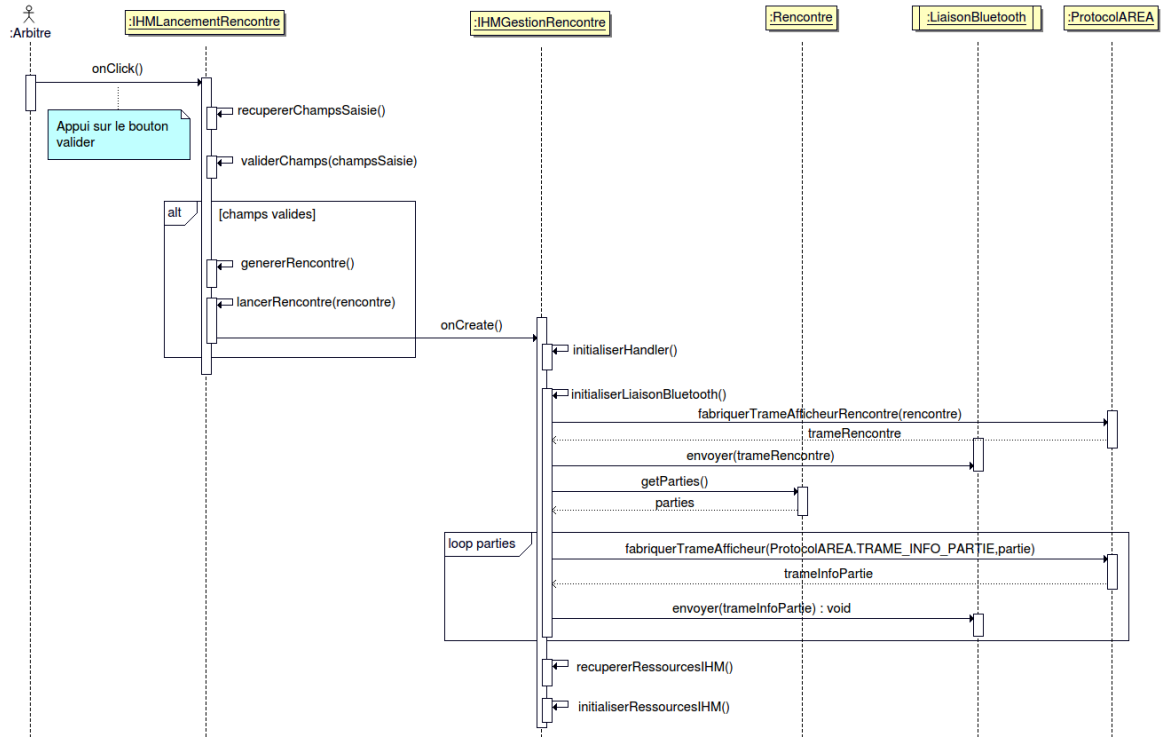
Dans cette méthode run() on vérifie dans un premier temps que la socket est connectée, si elle l'est on écrit les données convertit en octet sur le flux d'envoi. Cette partie de code se trouve dans un bloc try catch afin de pouvoir gérer les exceptions possiblement levées par la méthode write() de la classe OutputStream.

On finit par démarrer le Thread en appelant la méthode start() de la classe Thread.

Voici le diagramme de classe du scénario :



4.12.3 Démarrer une rencontre



Pour démarrer une rencontre l'arbitre appuie sur le bouton "Valider" de l'activité IHMLancementRencontre. Lorsque ce dernier est appuyé, la méthode onClick() est appelée.

On commence donc par récupérer les différentes valeurs qui ont été saisies à l'aide de la méthode recupererChampsSaisie() puis la méthode validerChamps() va vérifier que les champs sont tous valides, s' ils ne le sont pas une erreur est ajoutée sur le champ.

Si les champs sont valides alors on génère la rencontre puis on la lance grâce aux méthodes genererRencontre() et lancerRencontre().

La méthode lancerRencontre() démarre l'activité IHMGestionRencontre, ce qui va déclencher l'appel de la méthode onCreate(), cette méthode va dans un premier temps initialiser le handler permettant la communication de l'activité avec les objets de la classe LiaisonBluetooth par l'appel de la méthode initialiserHandler() puis initialiser la liaison au module afficheur grâce à la méthode initialiserLiaisonBluetooth().

Cette méthode va par la suite fabriquer une trame de rencontre et l'envoyer au module afficheur, puis récupérer les parties générées précédemment afin de pouvoir fabriquer et envoyer une trame d'information de partie pour chacune d'entre elles.

Enfin la méthode onCreate() se termine par la récupération et l'initialisation des ressources de l'IHM.

Définition de la méthode initialiserHandler() de la classe IHMGestionRencontre :

```
private void initialiserHandler()
{
   this.handler = new Handler(this.getMainLooper())
   {
       @Override
       public void handleMessage(@NonNull Message message)
       {
           Log.d(TAG, "[Handler] id du message = " + message.what);
           Log.d(TAG, "[Handler] contenu du message = " + message.obj.toString());

           switch (message.what)
           {
               case LiaisonBluetooth.CREATION_SOCKET:
                   Log.d(TAG, "[Handler] CREATION_SOCKET = " + message.obj.toString());
                   break;
               case LiaisonBluetooth.CONNEXION_SOCKET:
                   Log.d(TAG, "[Handler] CONNEXION_SOCKET = " + message.obj.toString());
                   liaisonModuleAfficheur.envoyer(
                       ProtocolAREA.fabriquerTrameAfficheurRencontre(rencontre));
                   envoyerPartiesAfficheur();
                   break;
               case LiaisonBluetooth.DECONNEXION_SOCKET:
                   Log.d(TAG, "[Handler] DECONNEXION_SOCKET = " + message.obj.toString());
                   break;
               case LiaisonBluetooth.RECEPTION_TRAME:
                   Log.d(TAG, "[Handler] RECEPTION_TRAME = " + message.obj.toString());
                   break;
           }
       }
   };
}
```

La méthode initialiserHandler() est assez représentative de la manière dont les activités communiquent avec la classe LiaisonBluetooth.

La classe LiaisonBluetooth envoie des messages via le handler, un message est constitué de plusieurs parties :

- Une partie permettant de l'identifier se traduisant par attribut par un attribut public 'what'

- Une partie qui représente le contenu du message et qui se traduit par un attribut public 'obj'

```
public static final int CREATION_SOCKET = 1;
public static final int CONNEXION_SOCKET = 2;
public static final int DECONNEXION_SOCKET = 3;
public static final int RECEPTION_TRAME = 4;
```

Plusieurs constantes sont déclarées afin de permettre d'identifier les différents types de messages, ils correspondent aux différents événements qui peuvent se produire lors du cycle de vie d'un objet de la classe LiaisonBluetooth.

La méthode handleMessage permet de gérer la réception de ces messages, cette dernière est essentiellement constituée d'un switch sur l'identifiant du message afin de pouvoir traiter chaque événement de manière différente.

Dans notre exemple, le cas traité est celui de la connexion du socket, lors de celle-ci une trame permettant d'envoyer les informations d'une rencontre est envoyée au module afficheur, puis la méthodes envoyerPartiesAfficheur() envoie une trame d'informations de partie pour chacune des parties de la rencontre.

Voici le diagramme de classe de ce scénario :

4.13 Tests de validation

Test	(input)	Résultat attendu	Validation
Affichage de net	Réception d'une trame NET	Message "NET" au centre de l'IHM	✓
Ajout d'un point a joueur	Appuie sur le bouton +1	Incrémentation du score	✓
Ajout d'une manche gagnée à un joueur	Terminer une manche	Incrémentation du nombre de manche gagnées	✓
Ajout d'un point à une équipe	Terminer une partie	Incrémentation du score d'une équipe	✓
Communication avec le module Afficheur_AREA	Ajout d'un point pour un joueur	Mise à jour de l'afficheur	✓
Afficher qui est le serveur et qui est le relanceur	Ajout d'un point pour un joueur	Affichage d'une icône à côté de son nom	✓
Editer les informations d'une rencontre	Saisie des informations	Affichage des informations dans la page de gestion de rencontre	✓
Sauvegarder les information d'une rencontre dans une base de données	Saisie des informations	Affichage de suggestions lors de la saisie	✗
Démarrer une rencontre	Appui sur le bouton "Valider" du menu d'édition de rencontre	Passage au menu de gestion de rencontre et mise à jour de l'afficheur	✓