ECRAN-TTPA [v1.1]

Table Tennis Performance Analyser

VIDAL Damien, GRENOD Pierre, SMANIOTTO Nathan, RACAMOND Adrien

Revue finale

11/06/2018

RACAMOND Adrien

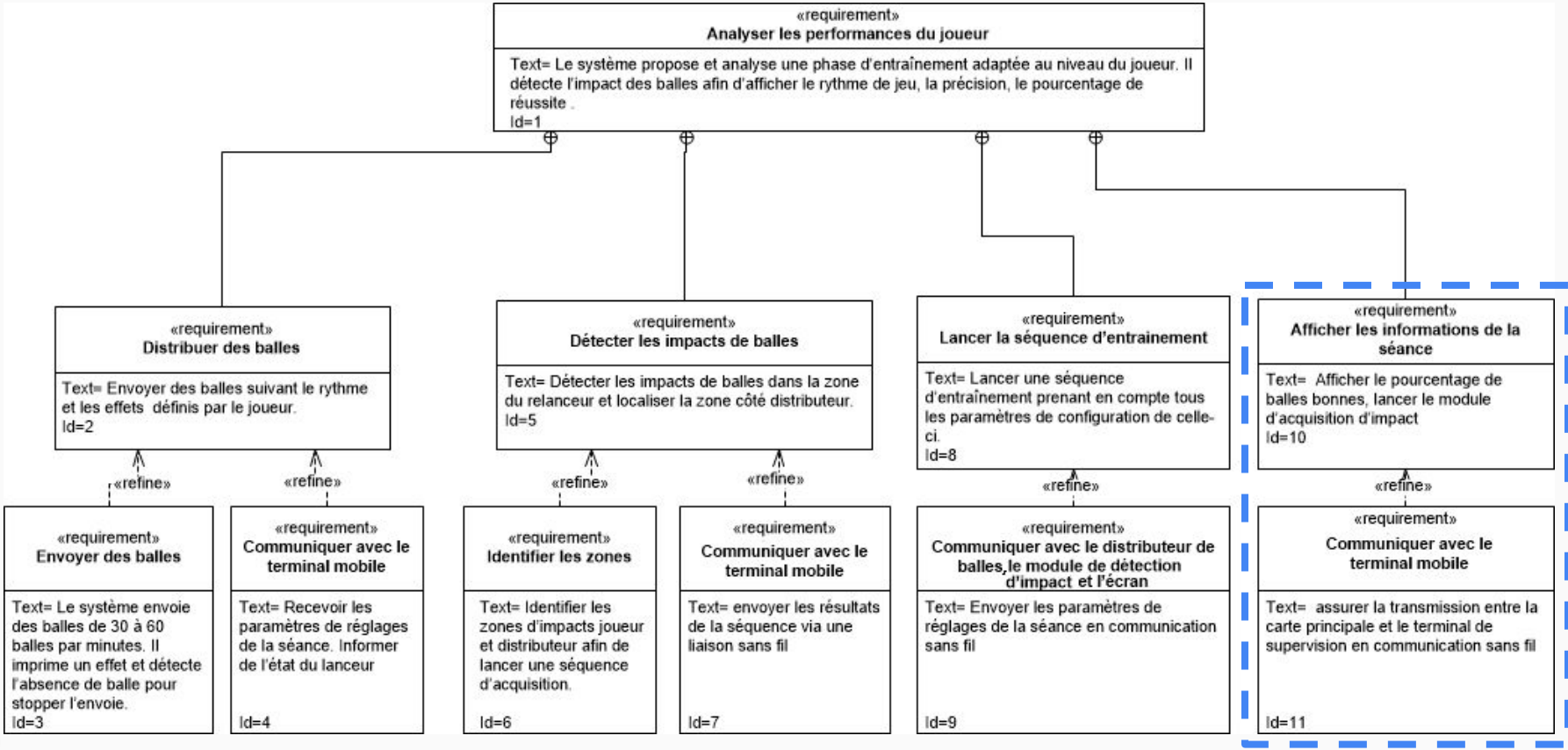
PRÉSENTATION GÉNÉRALE

Besoin Initial

100%

- Configurer et Lancer des séances d'entraînement automatisées
- Pouvoir visualiser en temps réel les performances des joueurs

Exigences


«requirement»
Analyser les performances du joueur
Text= Le système propose et analyse une phase d'entraînement adaptée au niveau du joueur. Il détecte l'impact des balles afin d'afficher le rythme de jeu, la précision, le pourcentage de réussite .
Id=1
«requirement»
Distribuer des balles
Text= Envoyer des balles suivant le rythme et les effets définis par le joueur.
Id=2
«requirement»
Détecter les impacts de balles
Text= Détecter les impacts de balles dans la zone du relanceur et localiser la zone côté distributeur.
Id=5
«requirement»
Lancer la séquence d'entrainement
Text= Lancer une séquence d'entraînement prenant en compte tous les paramètres de configuration de celle-ci.
Id=8
«requirement»
Afficher les informations de la séance
Text= Afficher le pourcentage de balles bonnes, lancer le module d'acquisition d'impact
Id=10
«refine»
«refine»
«refine»
«refine»
«refine»
«refine»
«requirement»
Envoyer des balles
Text= Le système envoie des balles de 30 à 60 balles par minutes. Il imprime un effet et détecte l'absence de balle pour stopper l'envoie.
Id=3
«requirement»
Communiquer avec le terminal mobile
Text= Recevoir les paramètres de réglages de la séance. Informer de l'état du lanceur
Id=4
«requirement»
Identifier les zones
Text= Identifier les zones d'impacts joueur et distributeur afin de lancer une séquence d'acquisition.
Id=6
«requirement»
Communiquer avec le terminal mobile
Text= envoyer les résultats de la séquence via une liaison sans fil
Id=7
«requirement»
Communiquer avec le distributeur de balles,le module de détection d'impact et l'écran
Text= Envoyer les paramètres de réglages de la séance en communication sans fil
Id=9
«requirement»
Communiquer avec le terminal mobile
Text= assurer la transmission entre la carte principale et le terminal de supervision en communication sans fil
Id=11

Exigences

1	2	3
4	5	6
7	8	9
	0	

Réponses aux exigences

Distribuer des balles

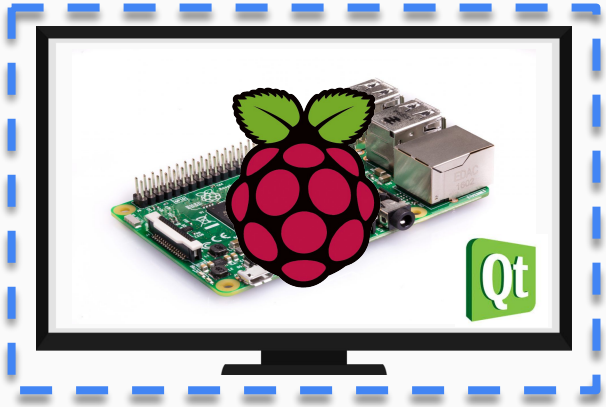
ARDUINO

ARDUINO

Détecter les impacts de balles

x10

Afficher les informations de la séance



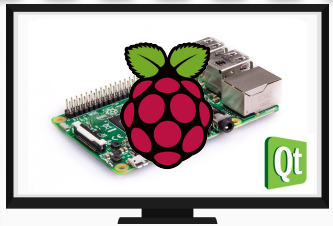
Paramétrer et lancer la séance d'entraînement

Répartition des tâches

Damien VIDAL
(EC)

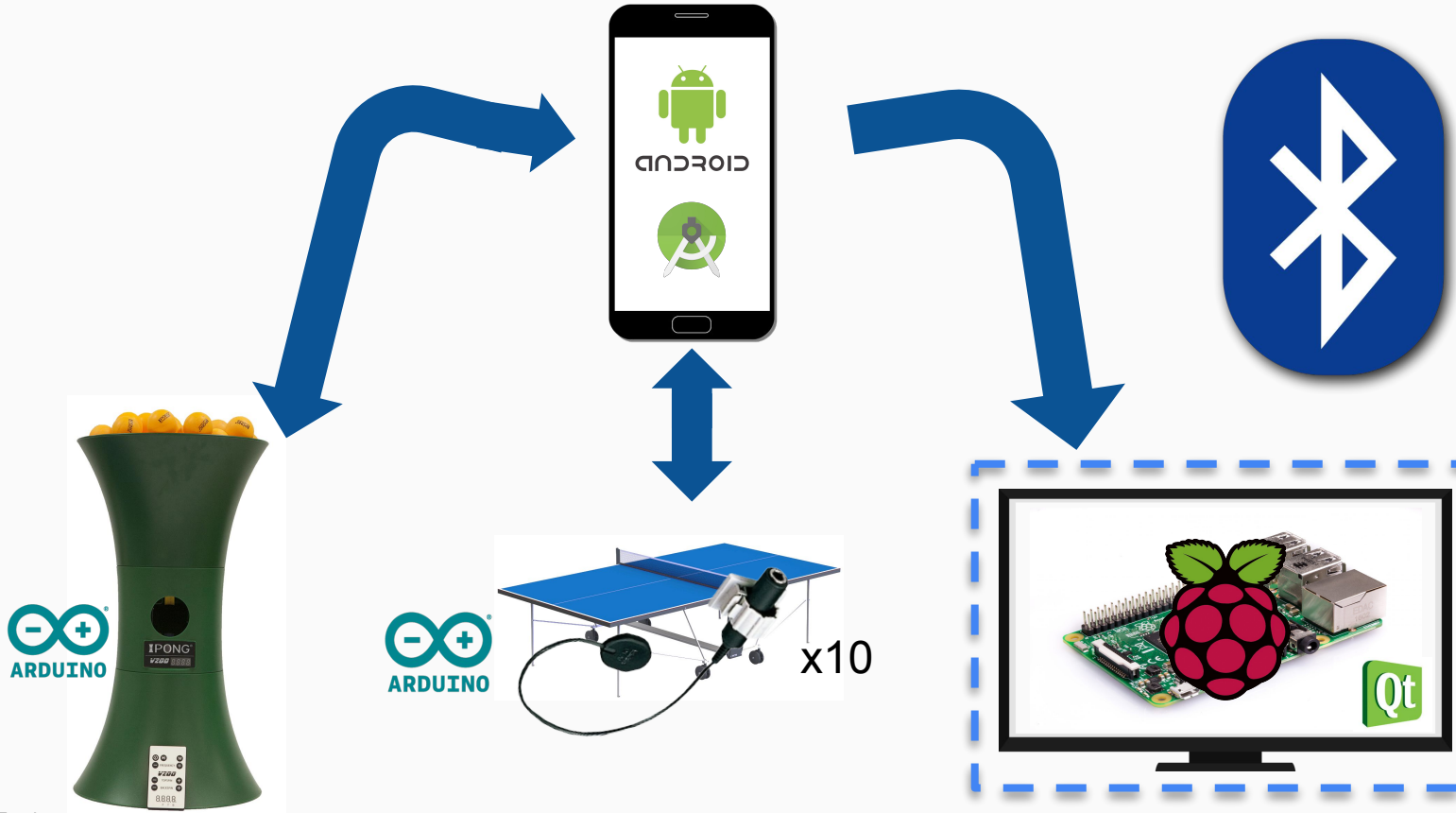
Nathan SMANIOTTO
(IR)

Pierre GRENOD
(EC)

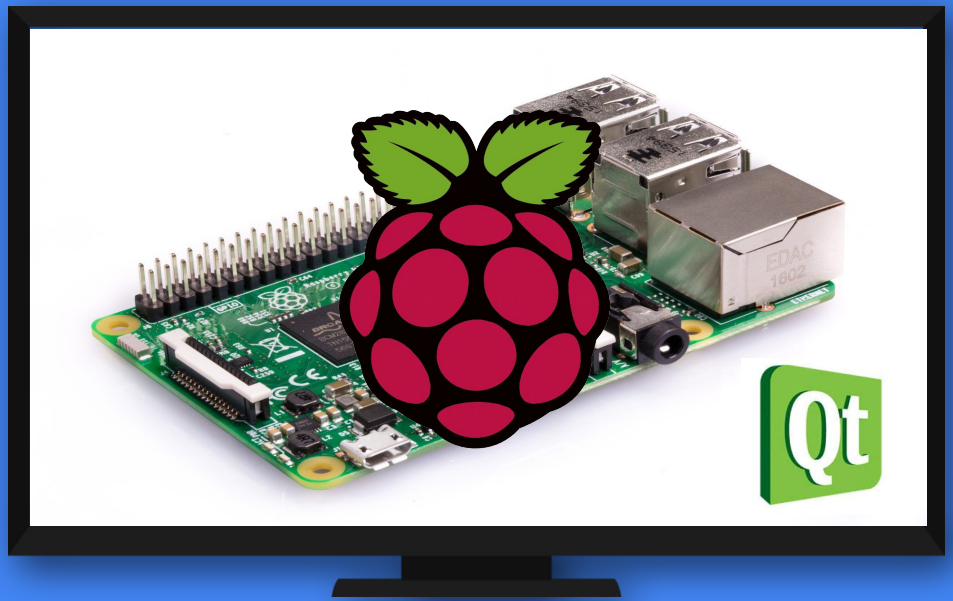


Adrien RACAMOND
(IR)

Communication entre les modules



ECRAN-TTPA



Introduction

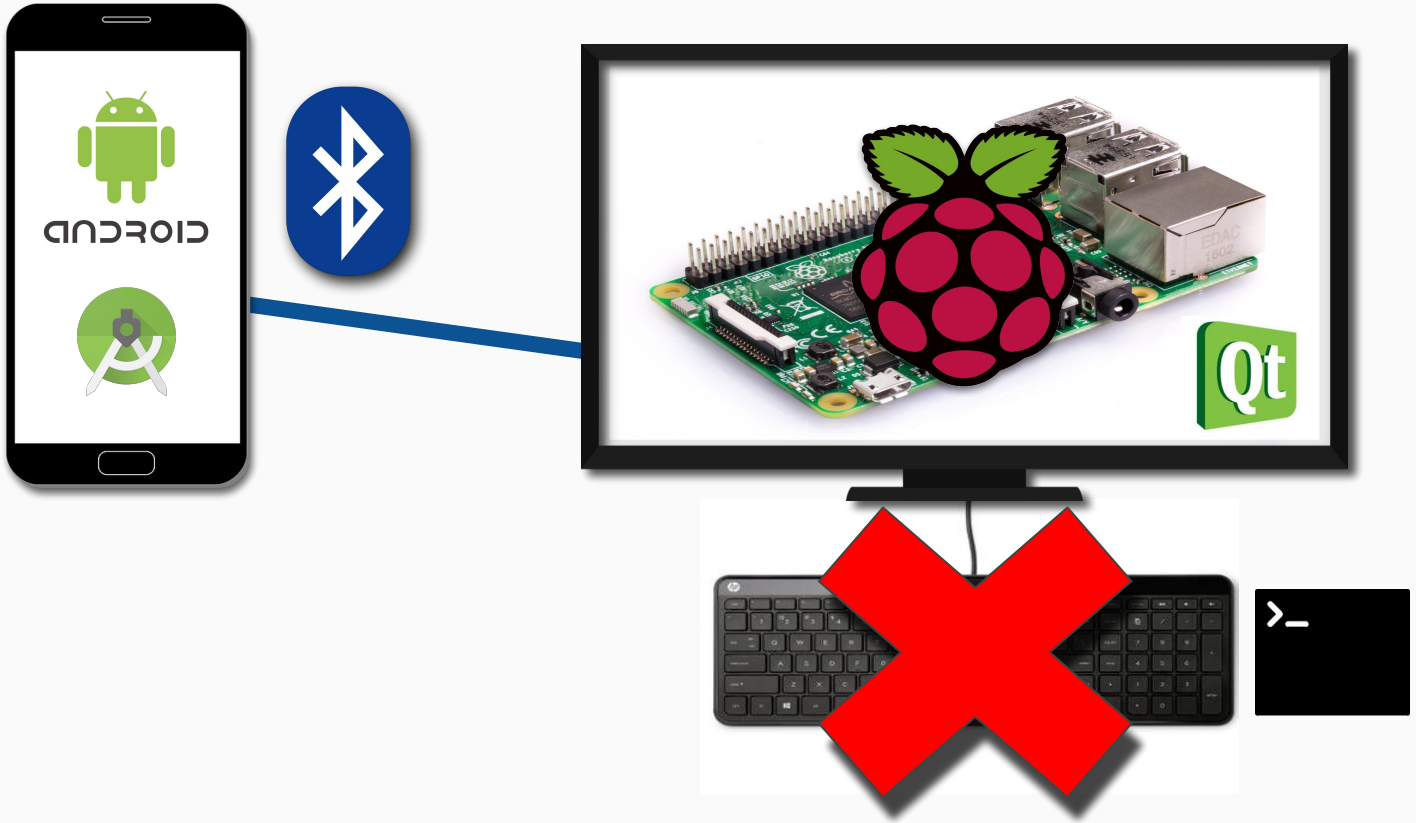
Ressources de développement

Développement	
Système d'exploitation du poste de développement	Ubuntu 12.04 LTS
Planification	Planner 0.14.5
Diagrammes UML	BOUML 4.23
Gestion de versions	Subversion 1.6.17
Langage utilisé	C++ avec le framework **Qt 4.8**
Documentation de code	Doxygen 1.7.6.1

Spécifications : Raspberry Pi 3

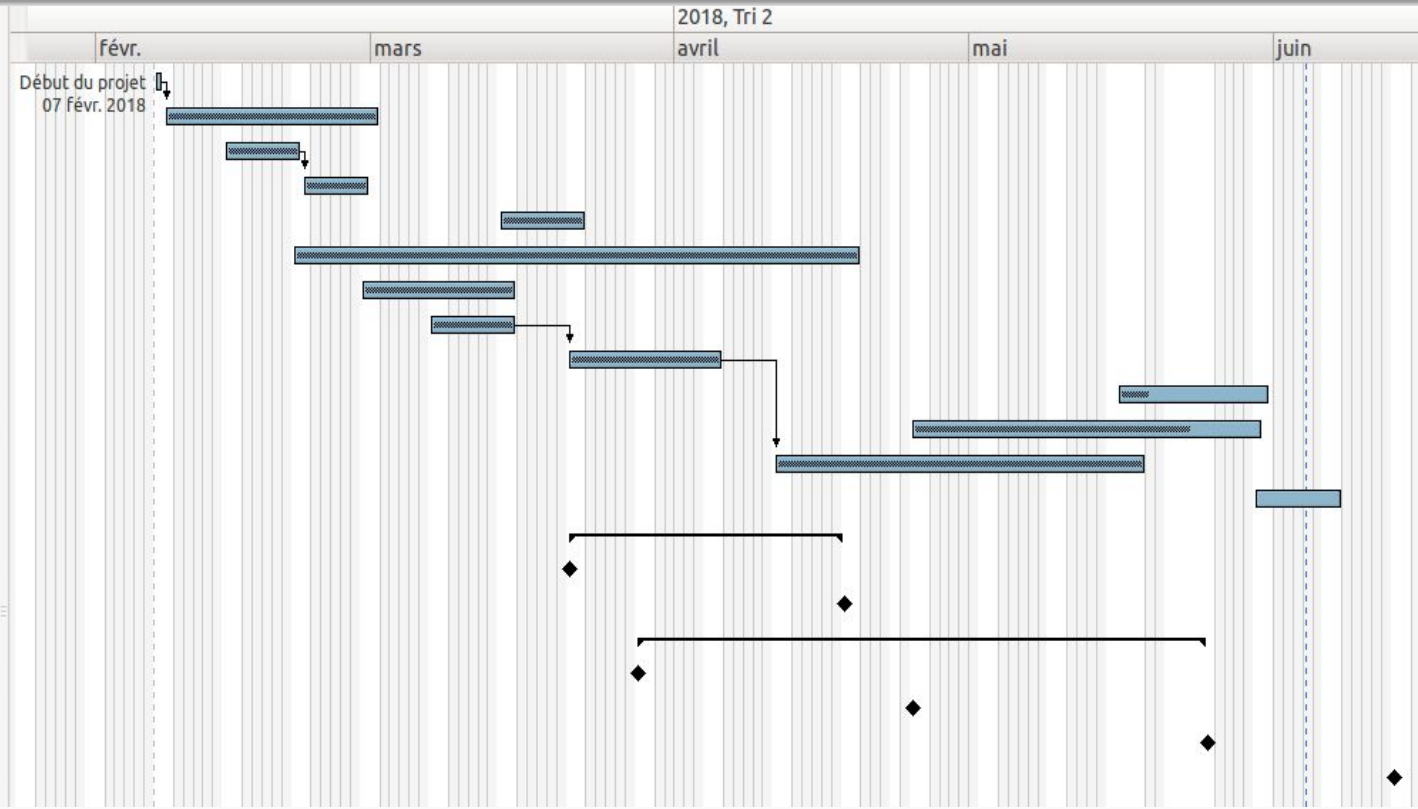
Raspberry Pi 3 Model B	
Système d'exploitation	**Raspbian** GNU/Linux 9.1 (stretch)
Processeur	Quad Core 1.2GHz Broadcom BCM2837 **ARMv8** CPU
RAM	**1GB**
Stockage	**Micro SD**
Sans-Fil	BCM43438 LAN et "**Bluetooth** Low Energy" (BLE)
Connectiques	40-pin GPIO, 4 ports USB 2, **HDMI**, CSI, DSI

Contrôle du module



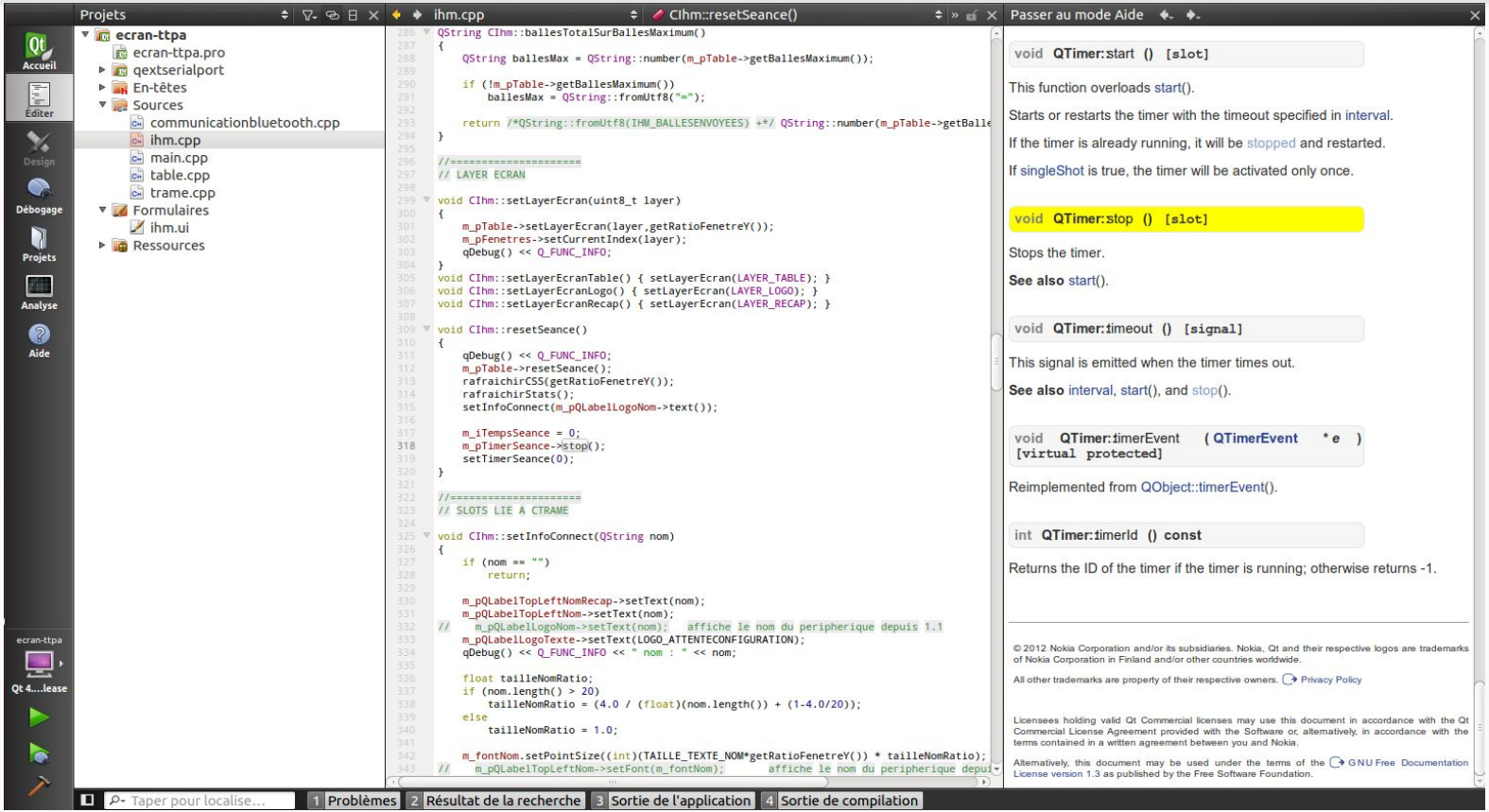
Planification

TPÉ	Nom	Trava
1	Planification	1j
2	Coder base Interface et Table	7j
3	Coder statistiques Zones %	3j
4	Ajouter Robot + Zone Objectif	2j
5	Coder base Ecran d'acceuil	4j
6	Developper Interface	18j
7	Ajouter Bluetooth	6j
8	Protocole	4j
9	Developper Comm BT / Ecran	6j
10	Ajout de fonctionalités	9j 3h
11	Ajout de statistiques récapitulatifs	16j
12	Redaction intensive du dossier	15j
13	Préparation demonstration	4j
14	**{Vérifications}**	
14.1	Vérifier + Reorganiser Code	N/D
14.2	Vérifier Protocole	N/D
15	**{Revues}**	
15.1	Revue	N/D
15.2	Revue	N/D
15.3	Rendu du dossier	N/D
16	Revue Finale	N/D



Utilisation de Qt dans le projet

4.8.1



Utilisation de Qt dans le projet

4.8.1

Objet1

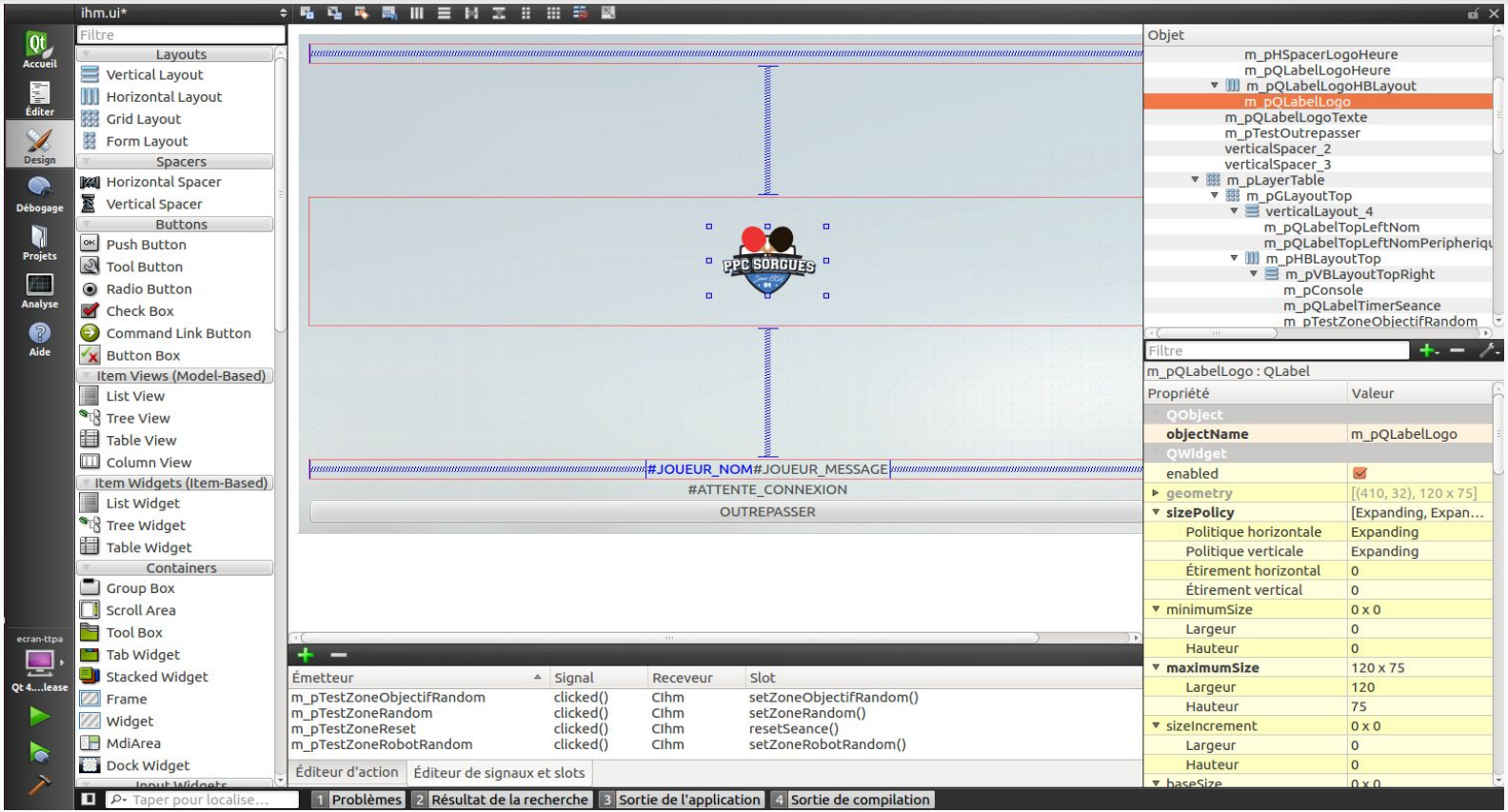
```
public slot:
    void methodeObjet1();

//----------------------------

connect( Objet2,
         SIGNAL( actionObjet2()),
         this,
         SLOT( methodeObjet1())
        );
```

Objet2

```
signals:
    void actionObjet2();

//----------------------------

emit actionObjet2();
```

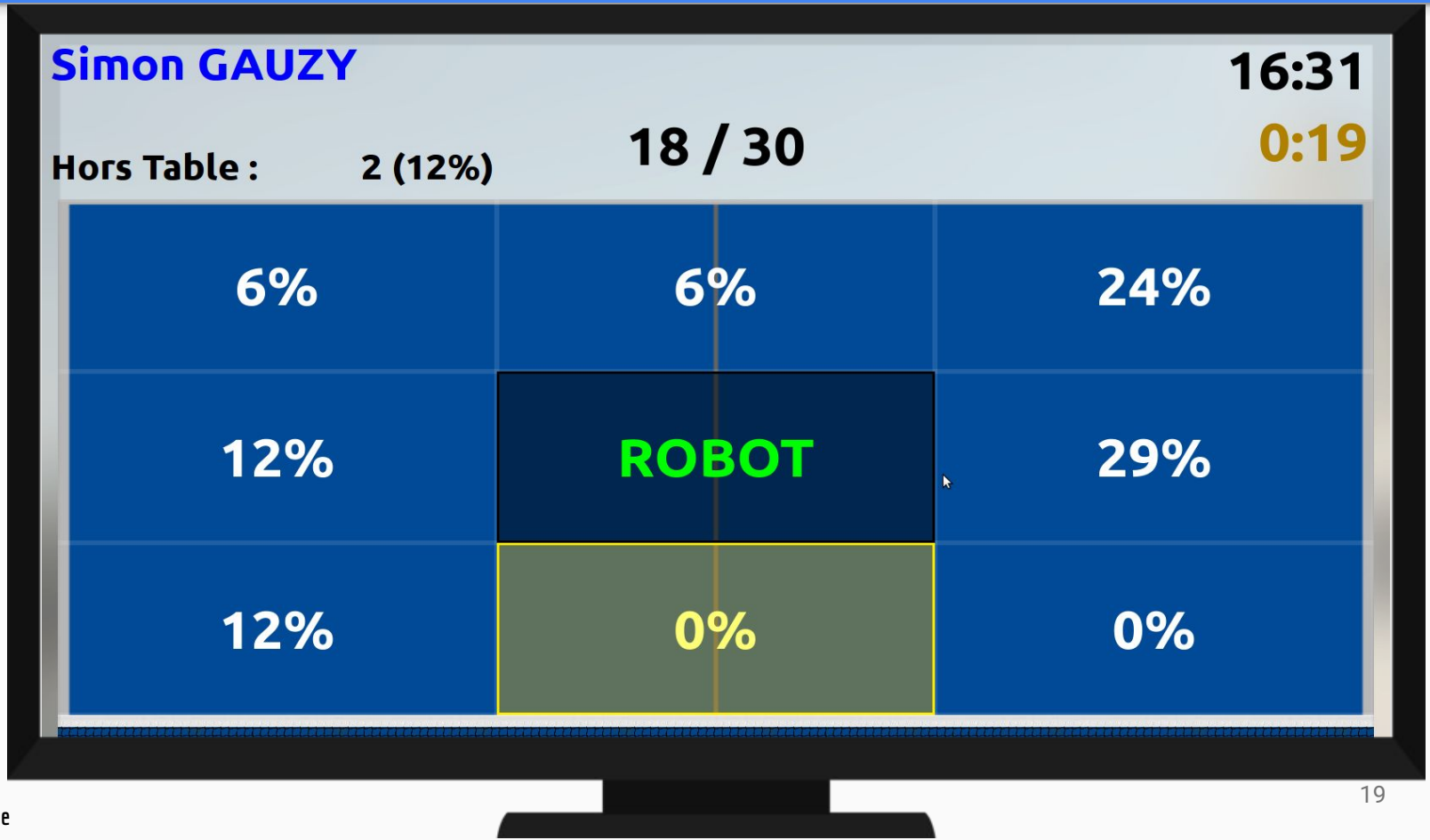
Utilisation de Qt dans le projet

4.8.1

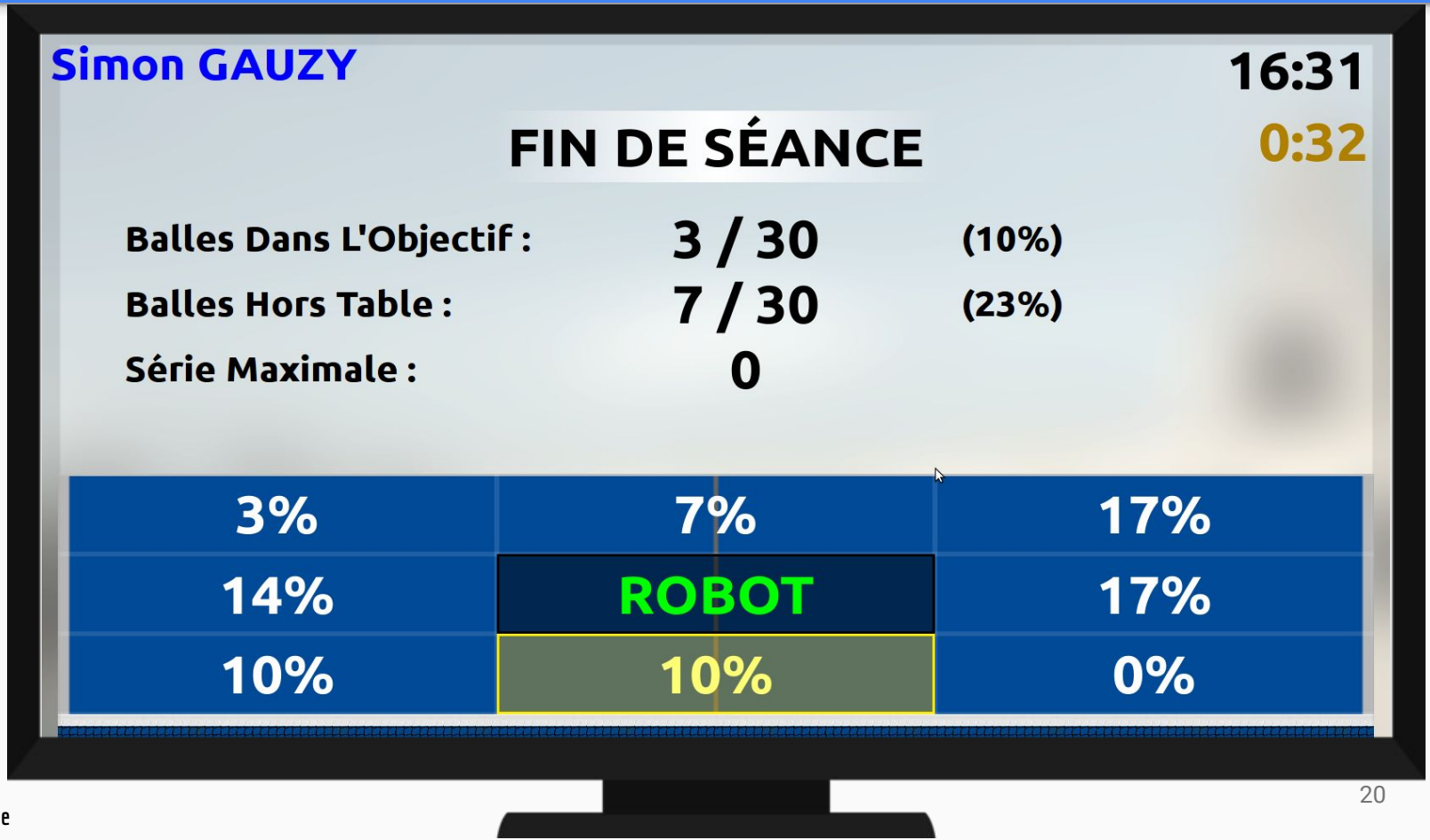


IHM : Écran de connexion

IHM : Écran de séance



IHM : Écran récapitulatif



Protocole TTPA

Entête protocole du système TTPA

Type commande, interprété par le programme et relié à une fonction correspondante

```
$TTPA:TYPE:{PARAMÈTRE}
```

Délimiteur de champs

Paramètre lié au type, il peut en avoir plusieurs en fonction du type, voir aucun

Protocole TTPA : Commandes (ÉCRAN)

CONNECT:NOM

SETSEANCE:R:O:B

IMPACT:Z

START

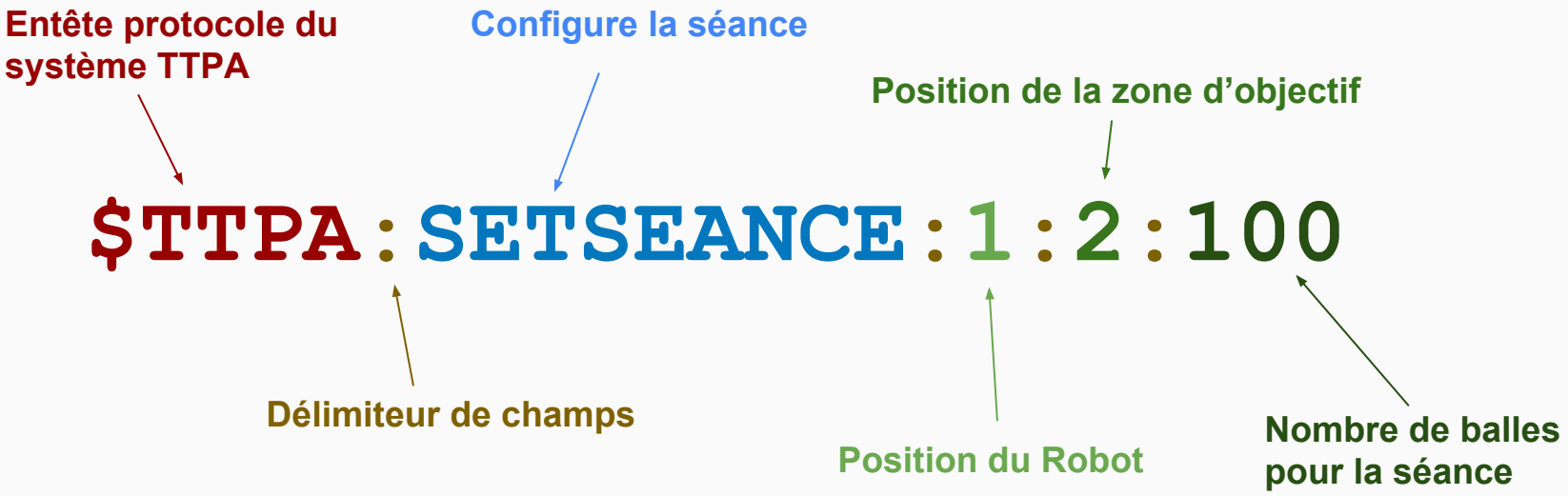
PAUSE

RESUME

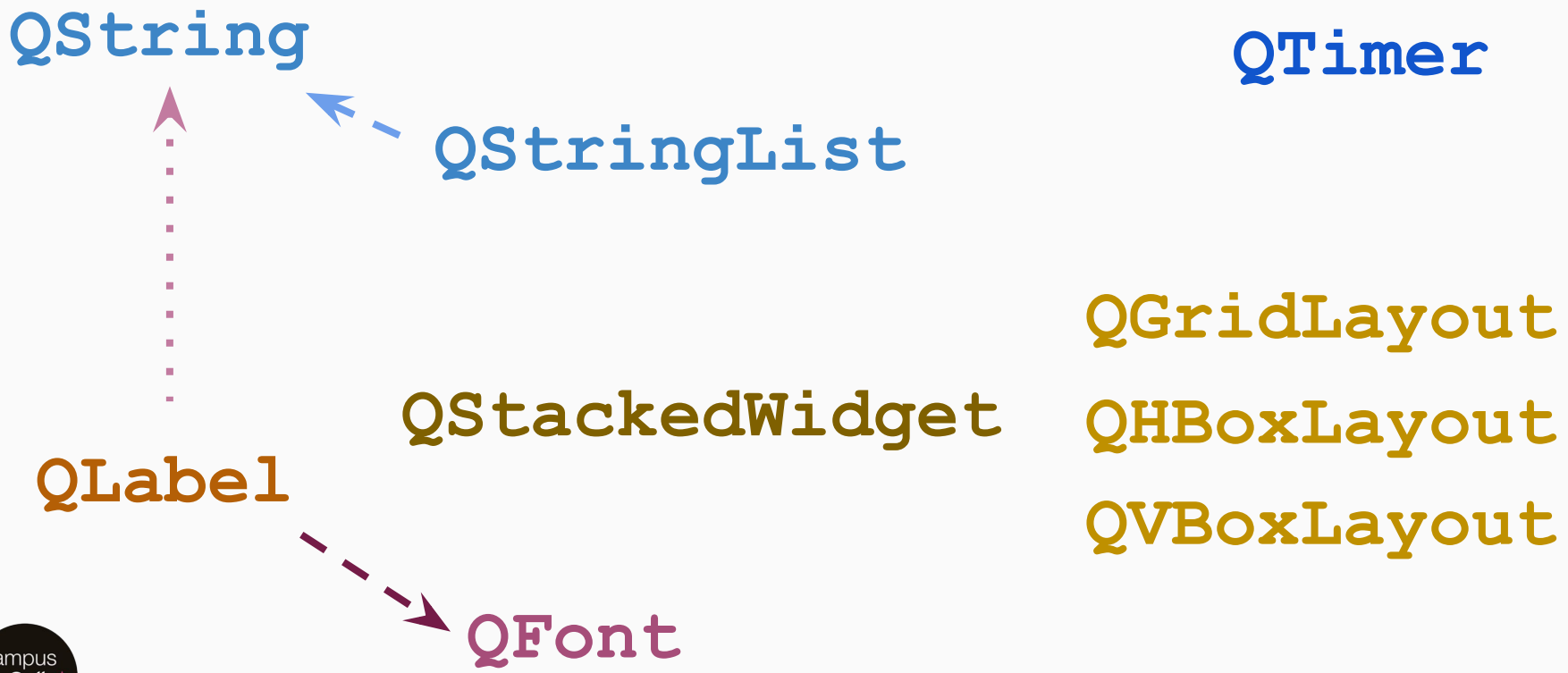
FINSEANCE

RESET

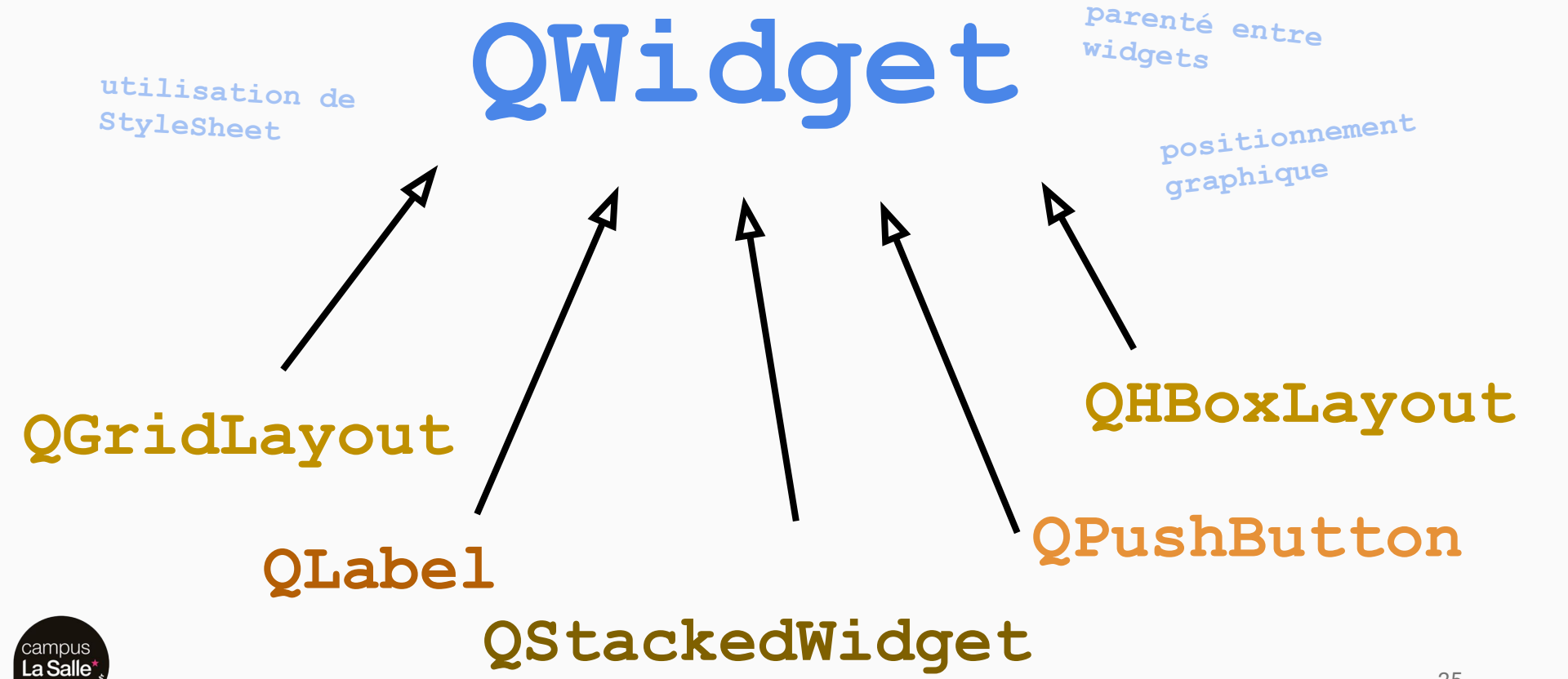
Protocole TTPA : Exemple



Utilisation des classes Qt



Utilisation des classes Qt : QWidget



Utilisation des classes Qt : QTimer

QTimer

Public Slots

void	**start**(int *msec*)
void	**start**()
void	**stop**()

› 1 public slot inherited from QObject

Signals

void	**timeout**()

› 2 signals inherited from QObject

Public Functions

	QTimer(QObject **parent* = nullptr)
virtual	**~QTimer**()
int	**interval**() const
std::chrono::milliseconds	**intervalAsDuration**() const
bool	**isActive**() const
bool	**isSingleShot**() const
int	**remainingTime**() const
std::chrono::milliseconds	**remainingTimeAsDuration**() const
void	**setInterval**(int *msec*)
void	**setInterval**(std::chrono::milliseconds *value*)
void	**setSingleShot**(bool *singleShot*)
void	**setTimerType**(Qt::TimerType *atype*)
void	**start**(std::chrono::milliseconds *msec*)
int	**timerId**() const
Qt::TimerType	**timerType**() const

› 32 public functions inherited from QObject

http://doc.qt.io/qt-5/qtimer.html

QTimer

Example for a one second (1000 millisecond) timer (from the Analog Clock example):

```
QTimer *timer = new QTimer(this);
connect(timer, SIGNAL(timeout()), this, SLOT(update()));
timer->start(1000);
```

From then on, the `update()` slot is called every second.

You can set a timer to time out only once by calling setSingleShot(true). You can also use the static QTimer::singleShot() function to call a slot after a specified interval:

```
QTimer::singleShot(200, this, SLOT(updateCaption()));
```

In multithreaded applications, you can use QTimer in any thread that has an event loop. To start an event loop from a non-GUI thread, use QThread::exec(). Qt uses the timer's thread affinity to determine which thread will emit the timeout() signal. Because of this, you

http://doc.qt.io/qt-5/qtimer.html

Utilisation des classes Qt : QTimer

QTimer : Classe pouvant gérer des temporisations

Besoins :

- Définir la durée de la temporisation ➜ **`setInterval(int)`**
- Démarrer le timer ➜ **`start([int])`**
- Arrêter le timer ➜ **`stop()`**
- Récupérer le signal de fin de temporisation ➜ **`timeout()`**

Utilisation des classes Qt : QTimer

QTimer

```
// CIhm.cpp

        m_pTimerSeance = new QTimer(this);
        m_pTimerSeance->setInterval(1000);
        m_iTempsSeance = 0;
        setTimerSeance(0);
        connect(m_pTimerSeance, SIGNAL(timeout()),this, SLOT(rafraichirTimerSeance()));
        // [. . .]
        m_pTimerSeance->start();
```

```
// CTable.cpp

        QTimer::singleShot(DELAI_COUP, this, SLOT(rafraichirInactif()));
```

Utilisation des classes Qt : QString

`QString` : Classe Qt pour gérer une chaîne de caractères

Besoins :

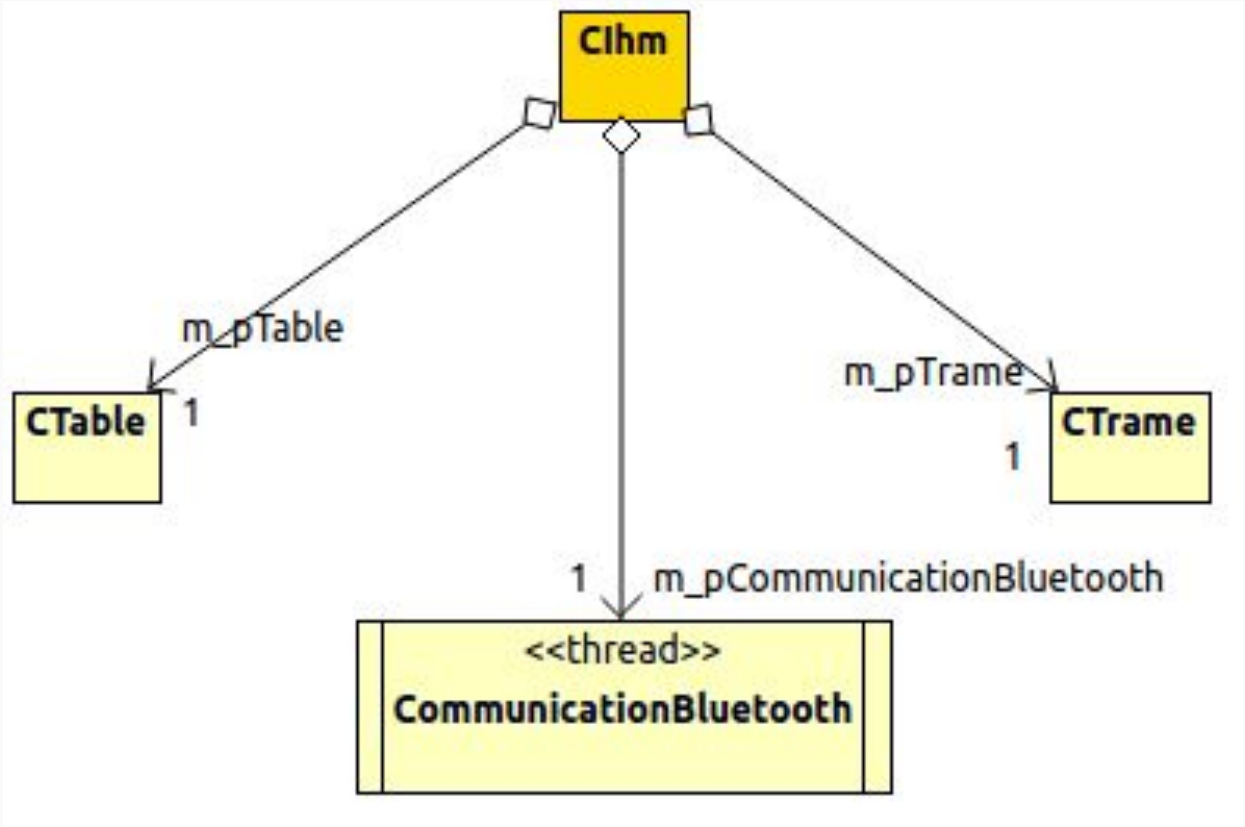
- Lire la longueur de la chaîne ➜ **`length()`**
- Vérifier la présence de caractères en début de chaîne ➜ **`startsWith(`**`QString`**`)`**
- Vérifier la présence de caractères en fin de chaîne ➜ **`endsWith(`**`QString`**`)`**
- Tester si la chaîne est vide ➜ **`isEmpty()`**
- Découper la chaîne en fonction d'un délimiteur ➜ **`split()`**
- Récupérer une chaîne sans les espaces blancs (\t, \n, \v, \f, \r et " ") au début et à la fin ➜ **`trimmed()`**
- Lire le caractère de la chaîne situé à une position donnée ➜ **`at(`**`int`**`)`**

QString

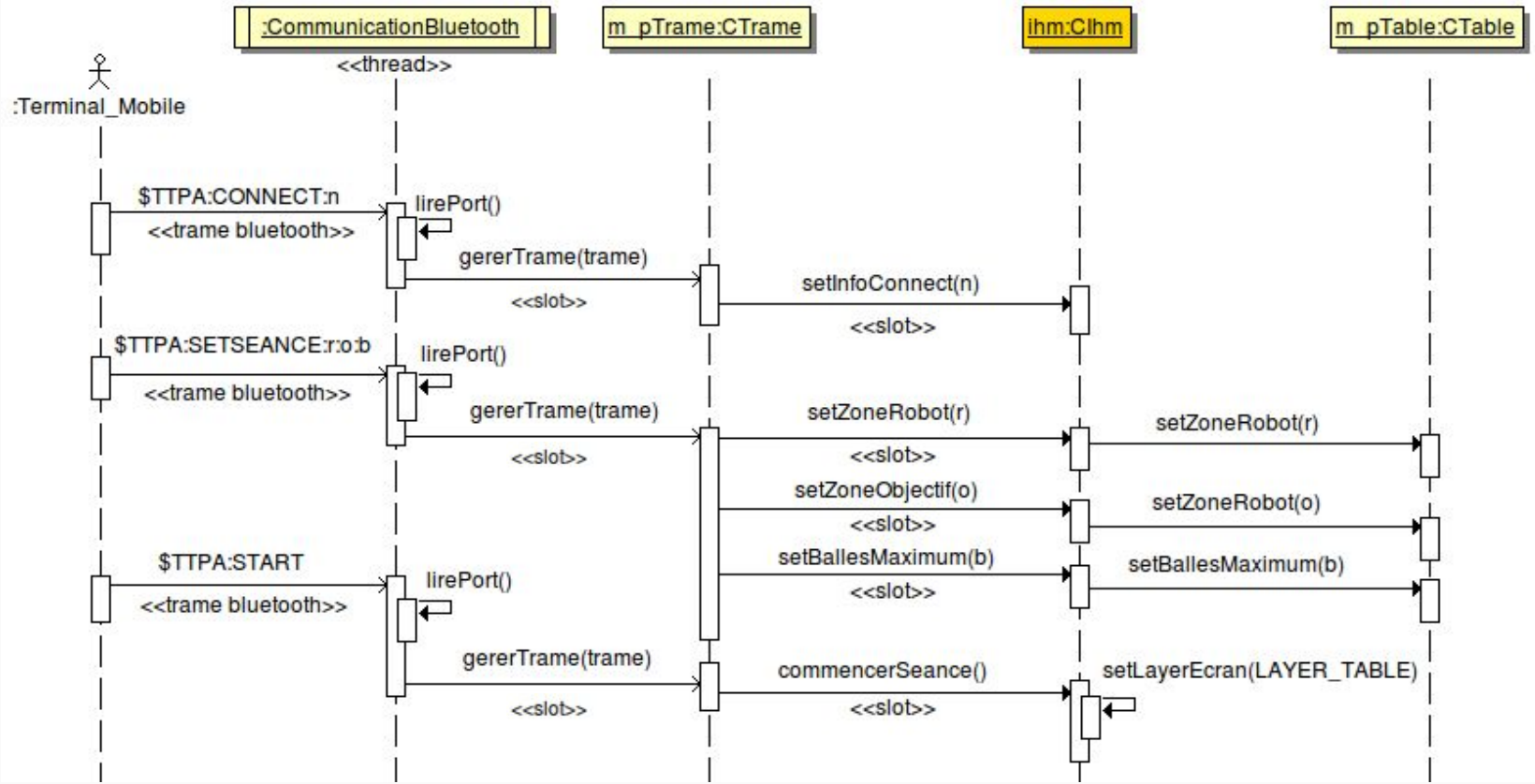
```
// CTrame.cpp
```

```
//$TTPA:SETSEANCE:[POS_ROBOT]:[POS_OBJECTIF]:[NB_BALLES_MAX]:[FREQ_ENVOI]
else if (getTrameLength(donneesRecues) == 5 && extraireElement(donneesRecues,1).startsWith("SETSEANCE"))
{
    if ((extraireElement(donneesRecues,2)).toInt() <= 0)    // AUCUN dans le cas d'un negatif ou zero
        emit setZoneRobot(ZONE_AUCUNE);
    else
        emit setZoneRobot((extraireElement(donneesRecues,2)).toInt() - 1);

    if ((extraireElement(donneesRecues,3)).toInt() <= 0)    // AUCUN dans le cas d'un negatif ou zero
        emit setZoneObjectif(ZONE_AUCUNE);
    else
        emit setZoneObjectif((extraireElement(donneesRecues,3)).toInt() - 1);

    emit setBallesMaximum((extraireElement(donneesRecues,4)).toInt());
}
```

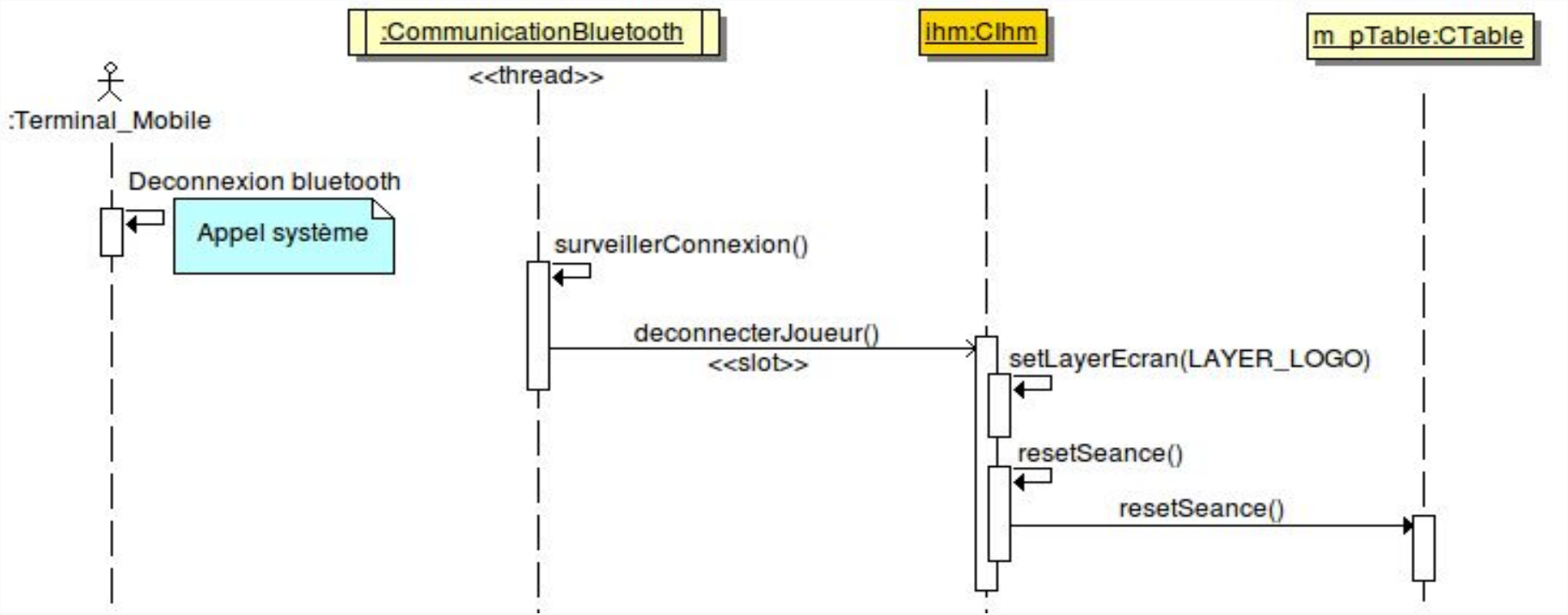
Diagramme de classes



Scénario : Configuration et démarrage



Scénario : Déconnexion du Bluetooth

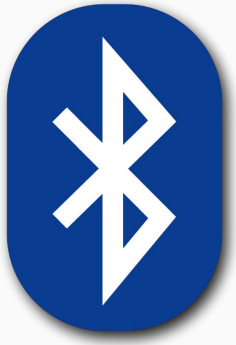


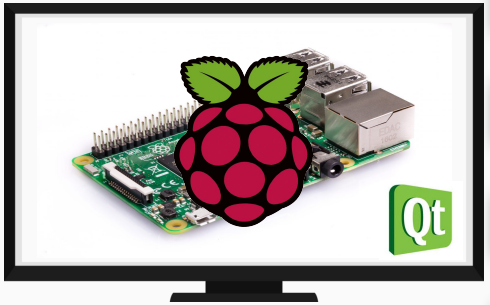
Tests de validation : version 1.1

Description	OUI	NON
Le système d'exploitation est installé et fonctionnel	X	
L'écran est configuré en mode "kiosque"	X	
La zone d'impact est identifiée et affichée en temps réel	X	
Les données de la séance (le pourcentage de balles par zones et nombre de pourcentages de balles bonnes) sont affichées en temps réel	X	
Les liaisons sans fil sont opérationnelles	X	
Les informations sont affichées en fin de séquence	X	

CONCLUSION

Communication avec Mobile-TTPA





Scénario : Déroulement d'une séance