Projet Darts

version 0.2

BTS SNIR LaSalle Avignon 2020

Table des matières

1 Le projet

Le système **DARTS** est un système numérique permettant de jouer au jeu de fléchettes électroniques.

1.1 Table des matières

1.2 Informations

Auteur

Fabien Bounoir bounoirfabien@gmail.com

Date

2020

Version

0.2

Voir également

https://svn.riouxsvn.com/darts-2020/

2 Changelog

r1 | www-data | 2020-02-01 15 :03 :29 +0100 (sam. 01 févr. 2020) | 1 ligne

Creating initial repository structure

3 README

3.1 Projet

3.1.1 Présentation

Le système **DARTS** est un système numérique permettant de jouer au jeu de fléchettes électroniques.

Le système DARTS est décomposé en trois modules, dont deux modules sont réalisés par des étudiants IR :

— Module de gestion de partie (**Mobile-DARTS**) : les joueurs paramètrent et lancent la partie à partir d'une application sur un terminal mobile (sous Android) ;
— Module de détection des impacts (Cible-DARTS) : la cible est équipée de capteurs permettant d'identifier la zone impactée par les fléchettes envoyées par les joueurs ;
— Module de visualisation de partie (**Écran-DARTS**) : les joueurs, les arbitres et le public peuvent visualiser en “temps réel” le déroulement de la partie (nombre de manche, point restant dans la manche, moyenne des volées, ...) sur un écran de télévision.

3.1.2 Module de visualisation de partie (Écran-DARTS)

Ce module correspond à la partie "affichage" du système. Il a pour objectifs de réaliser la récupération d'informations envoyées par le terminal mobile, le calcul et l'affichage les statistiques pour la partie actuelle. Il communique en Bluetooth uniquement avec le terminal mobile Android.

Sur l'écran, les joueurs pourront visualiser en continu :

- le nom des joueurs (si existant), la durée écoulée de la partie ;
- le type de jeu en cours, le score et le nombre de manches gagnées par chaque joueur
- la plus haute et la moyenne des volées de chaque joueur

Les données visualisées sont donc :

- Le type de jeu
- Le numéro de la manche
- Le score de la partie en cours
- La moyenne des volées

3.1.3 Informations

Auteur

Fabien Bounoir bounoirfabien@gmail.com

Date

2020

Version

0.2

Voir également

https://svn.riouxsvn.com/darts-2020/

4 A propos

Auteur

Fabien Bounoir bounoirfabien@gmail.com

Date

2020

Version

0.2

Voir également

https://svn.riouxsvn.com/darts-2020/

5 Licence GPL

6 Documentation des espaces de nommage

6.1 Référence de l'espace de nommage Ui

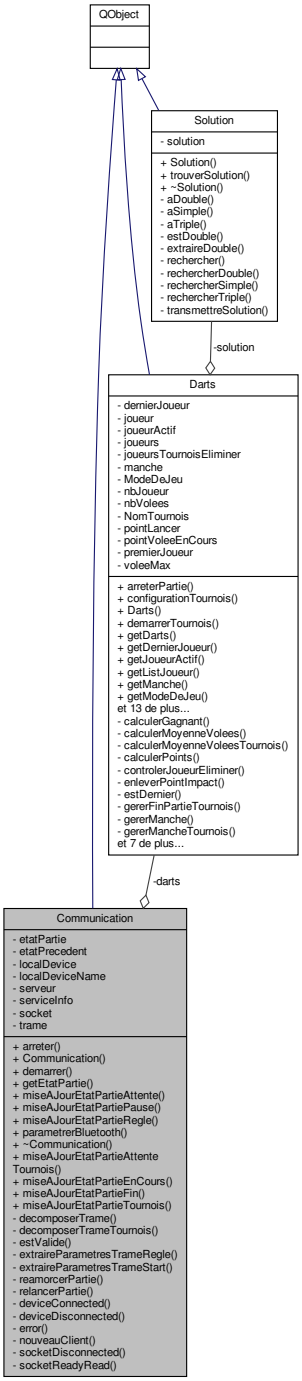
7 Documentation des classes

7.1 Référence de la classe Communication

Déclaration de la classe Communication via la liaison Bluetooth (Module Ecran-DARTS)

```
#include "communication.h"
```

Graphe de collaboration de Communication :



Types publics

— enum EtatPartie {
Attente = 0, EnCours = 1, Fin = 2, Pause = 3,
Regle = 4, AttenteTournois = 5, Tournois }
contient les different etat de la partie

Connecteurs publics

— void miseAJourEtatPartieAttenteTournois ()

Slot appelé pour mettre à jour l'état de la partie à attenteTournois.
- void miseAJourEtatPartieEnCours ()
  Slot appelé pour mettre à jour l'état de la partie à EnCours.
- void miseAJourEtatPartieFin ()
  Slot appelé pour mettre à jour l'état de la partie à Fin.
- void miseAJourEtatPartieTournois ()
  Slot appelé pour mettre à jour l'état de la partie à tournois.

Signaux

- void afficherAttenteConnexion ()
  signal émis quand un appareil se déconnecte
- void afficherRegle (QString regle)
  signal emit pour afficher les regles
- void appareilConnecter ()
  signal émis quand un nouvel appareil est connecté
- void erreurBluetooth (QString erreur)
  signal emit quand un probleme de configuration bluetooth est detecté
- void jouerSon (QString)
- void pause ()
  signal qui mettra en pause la partie
- void play ()
  signal qui relancera le chronometrage de la partie la partie
- void resetPartie ()
  signal qui reinitialisera la partie en cours
- void stopperRegle ()
  signal emit pour stopper la lecture des regles

Fonctions membres publiques

- void arreter ()
  Méthode qui arrête le serveur.
- Communication (Darts *darts, QObject *parent=nullptr)
  constructeur de la classe Communication
- void demarrer ()
  Méthode qui démarre le serveur.
- int getEtatPartie ()
  Méthode qui permet de recuperer l'etat de la partie.
- void miseAJourEtatPartieAttente ()
  Méthode appelé pour mettre à jour l'état de la partie à Attente.
- void miseAJourEtatPartiePause ()
  Méthode appelé pour mettre à jour l'état de la partie à Pause.
- void miseAJourEtatPartieRegle ()
  Méthode appelé pour mettre à jour l'état de la partie à regle.
- void parametrerBluetooth ()
  Méthode qui configure la connexion Bluetooth en mode serveur.
- ~Communication ()
  destructeur de la classe Communication

Connecteurs privés

- void deviceConnected (const QBluetoothAddress &adresse)
  Slot appelé quand un nouvel appareil est connecté
- void deviceDisconnected (const QBluetoothAddress &adresse)
  Slot appelé quand un appareil est déconnecté
- void error (QBluetoothLocalDevice : :Error erreur)
  Slot appelé quand il y a une erreur avec l'appareil bluetooth.
- void nouveauClient ()
  Slot appelé quand un nouveau client veut se connecter.
- void socketDisconnected ()
  Slot appelé quand la communication est deconnectée.
- void socketReadyRead ()
  Slot appelé quand une nouvelle trame est disponible.

Fonctions membres privées

- void decomposerTrame ()
  Méthode qui decompose la trame reçue.
- void decomposerTrameTournois ()
  Méthode qui extrait les paramètres de la trame TOURNOIS.
- bool estValide ()
  Méthode qui test si la trame est valide.
- void extraireParametresTrameRegle ()
  Méthode qui extrait les paramètres de la trame REGLE.
- void extraireParametresTrameStart (QStringList &joueurs, QString &modeJeu)
  Méthode qui extrait les paramètres de la trame START.
- void reamorcerPartie ()
  Méthode qui relance la partie.
- void relancerPartie ()
  Méthode qui gere l'initialisation d'une partie de flechette.

Attributs privés

- Darts * darts
  Association avec l'objet darts.
- EtatPartie etatPartie
  L'état de la partie.
- int etatPrecedent
  Contient l'etat dans lequel se trouver l'application.
- QBluetoothLocalDevice localDevice
  L'interface Bluetooth de la Raspberry Pi.
- QString localDeviceName
  Nom du peripherique local.
- QBluetoothServer * serveur
  Le serveur Bluetooth.
- QBluetoothServiceInfo serviceInfo
  Informations sur le service bluetooth.
- QBluetoothSocket * socket
  La socket de communication Bluetooth.
- QString trame
  La trame recue.

7.1.1 Description détaillée

Déclaration de la classe Communication via la liaison Bluetooth (Module Ecran-DARTS)

Cette classe s'occupe de la partie communication Bluetooth en mode serveur (configuration, réception et traitement des trames du protocole DART)

Définition à la ligne 41 du fichier communication.h.

7.1.2 Documentation des énumérations membres

7.1.2.1 EtatPartie

`enum Communication::EtatPartie`

contient les different etat de la partie

Valeurs énumérées

Attente	
EnCours	
Fin	
Pause	
Regle	

Définition à la ligne 61 du fichier communication.h.

```
00062     {
00063         Attente = 0,
00064         EnCours = 1,
00065         Fin = 2,
00066         Pause = 3,
00067         Regle = 4,      //État attente pendant la lecture des règles, seule une trame reset peut être
      reçu pour arrêter la lecture.
00068         AtteneTournois = 5,
00069         Tournois
00070     };
```

7.1.3 Documentation des constructeurs et destructeur

7.1.3.1 Communication()

```
Communication::Communication (
            Darts * darts,
            QObject * parent = nullptr )  [explicit]
```

constructeur de la classe Communication

Paramètres

darts	
parent	

Définition à la ligne 22 du fichier communication.cpp.

Références miseAJourEtatPartieAttente(), et parametrerBluetooth().

```
00022                                                   : QObject(parent),
      darts(darts), serveur(nullptr), socket(nullptr),
      localDeviceName("Ecran-Darts"), trame("")
00023 {
00024     qDebug() << Q_FUNC_INFO;
00025
00026     parametrerBluetooth();
00027
00028     miseAJourEtatPartieAttente();
00029 }
```

7.1.3.2 ~Communication()

```
Communication::~Communication ( )
```

destructeur de la classe Communication

Définition à la ligne 36 du fichier communication.cpp.

Références arreter(), et localDevice.

```
00037 {
00038     arreter();
00039     localDevice.setHostMode(QBluetoothLocalDevice::HostPoweredOff);
00040     qDebug() << Q_FUNC_INFO;
00041 }
```

7.1.4 Documentation des fonctions membres

7.1.4.1 afficherAttenteConnexion

```
void Communication::afficherAttenteConnexion ( )  [signal]
```

signal émis quand un appareil se déconnecte

Référencé par deviceDisconnected().

7.1.4.2 afficherRegle

```
void Communication::afficherRegle (
            QString regle )  [signal]
```

signal emit pour afficher les regles

Référencé par extraireParametresTrameRegle(), et extraireParametresTrameStart().

7.1.4.3 appareilConnecter

```
void Communication::appareilConnecter ( )  [signal]
```

signal émis quand un nouvel appareil est connecté

Référencé par deviceConnected().

7.1.4.4 arreter()

```
void Communication::arreter ( )
```

Méthode qui arrête le serveur.

arrete le serveur

Définition à la ligne 123 du fichier communication.cpp.

Références localDevice, serveur, serviceInfo, et socket.

Référencé par ~Communication().

```
00124 {
00125     if (!localDevice.isValid())
00126         return;
00127
00128     if (!serveur)
00129         return;
00130
00131     serviceInfo.unregisterService();
00132
00133     if (socket)
00134     {
00135         if (socket->isOpen())
00136         {
00137            socket->close();
00138         }
00139         delete socket;
00140         socket = nullptr;
00141     }
00142
00143     delete serveur;
00144     serveur = nullptr;
00145 }
```

7.1.4.5 decomposerTrame()

```
void Communication::decomposerTrame ( )  [private]
```

Méthode qui decompose la trame reçue.

Méthode qui decompose la trame reçue et signale l'état en fonction du type de trame. $DART ;START ;501 ;1 ;2 ;fabien ;erwan

$DART ;GAME ;3 ;7

$DART ;GAME ;3 ;7

$DART ;REGLE

$DART ;PAUSE

$DART ;PLAY

$DART ;STOP

$DART ;STOP

Définition à la ligne 190 du fichier communication.cpp.

Références Darts : :arreterPartie(), darts, decomposerTrameTournois(), DELIMITEUR_FIN, estValide(), etatPartie, etatPrecedent, extraireParametresTrameRegle(), extraireParametresTrameStart(), jouerSon(), miseAJourEtatPartiePause(), pause(), play(), reamorcerPartie(), Darts : :receptionnerImpact(), Darts : :receptionnerImpactTournois(), Darts : :reinitialiserPartie(), relancerPartie(), resetPartie(), stopperRegle(), et trame.

Référencé par socketReadyRead().

```
00191 {
00192     if(estValide()) //test si la trame est valide
00193     {
00194         trame.remove(DELIMITEUR_FIN);
00195         if(trame.contains("START") && (etatPartie == EtatPartie::Attente ||
      etatPartie == EtatPartie::Fin))
00196         {
00197             emit resetPartie();
00198             darts->reinitialiserPartie();
00199
00200             QString modeJeu;
00201             QStringList joueurs;
00202
00203             extraireParametresTrameStart(joueurs, modeJeu);
00204         }
00205         else if(trame.contains("GAME") && etatPartie == EtatPartie::EnCours)
00206         {
00207             darts->receptionnerImpact(trame.section(";",2,2).toInt(),
      trame.section(";",3,3).toInt());
00208         }
00209         else if(trame.contains("GAME") && etatPartie == EtatPartie::Tournois)
00210         {
00211             darts->receptionnerImpactTournois(
      trame.section(";",2,2).toInt(), trame.section(";",3,3).toInt());
00212         }
00213         else if(trame.contains("REGLE")&& etatPartie != EtatPartie::Regle)
00214         {
00215             extraireParametresTrameRegle();
00216         }
00217         else if(trame.contains("PAUSE") && (etatPartie == EtatPartie::EnCours ||
      etatPartie == EtatPartie::Tournois))
00218         {
00219             etatPrecedent = etatPartie;
00220             emit pause();
00221             miseAJourEtatPartiePause();
00222         }
00223         else if(trame.contains("PLAY") && etatPartie == EtatPartie::Pause)
00224         {
00225             emit play();
00226             relancerPartie();
00227         }
00228         else if(trame.contains("SON"))
00229         {
00230             emit jouerSon(trame.section(";",2,2));
```

```
00231         }
00232         else if(trame.contains("TOURNOIS"))
00233         {
00234             decomposerTrameTournois();
00235         }
00236         else if(trame.contains("RESET")) // quelque soit l'état de la partie    /** $DART;RESET */
00237         {
00238             reamorcerPartie();
00239         }
00240         else if(trame.contains("STOP") && (etatPartie == EtatPartie::EnCours ||
      etatPartie == EtatPartie::Pause))
00241         {
00242             darts->arreterPartie();
00243         }
00244         else if(trame.contains("STOP") && (etatPartie == EtatPartie::Regle))   //permet
       l'arret des regles en cours de lecture
00245         {
00246             emit stopperRegle();
00247         }
00248         else
00249         {
00250             qDebug() << Q_FUNC_INFO << "Trame non Traité: " << trame;
00251         }
00252     }
00253 }
```

7.1.4.6 decomposerTrameTournois()

```
void Communication::decomposerTrameTournois ( )  [private]
```

Méthode qui extrait les paramètres de la trame TOURNOIS.

Methode qui decompose la trame TOURNOIS. $DART ;TOURNOIS ;CONFIG ;501 ;Tournois inter classe ;4 ;Fabien ;Thierry ;Erwan ;Julia

$DART ;TOURNOIS ;PLAY

Définition à la ligne 317 du fichier communication.cpp.

Références Darts : :configurationTournois(), darts, Darts : :demarrerTournois(), etatPartie, Darts : :reinitialiserPartie(), et trame.

Référencé par decomposerTrame().

```
00318 {
00319     QString modeJeu;
00320     QString nomTournois;
00321     QStringList joueurs;
00322
00323     if(trame.contains("CONFIG") && (etatPartie == EtatPartie::Attente ||
      etatPartie == EtatPartie::Fin))
00324     {
00325         modeJeu = trame.section(";",3,3);
00326
00327         if(trame.section(";",5,5).toInt()%2 !=0)
00328         {
00329             qDebug() << "Nombre de joueur insuffissant";
00330             return;
00331         }
00332
00333         darts->reinitialiserPartie();
00334
00335         for(int i = 0;i <= trame.section(";",5,5).toInt();i++)  //boucle qui recuperer les noms des
       differents joueurs
00336         {
00337             if(trame.section(";",5+i,5+i) == "")    //test si le joueur a un nom
00338             {
00339                 joueurs.push_back("Joueur[" + QString::number(i) + "]");
00340             }
00341             else
00342             {
00343                 joueurs.push_back(trame.section(";",5+i,5+i));
00344             }
00345         }
00346         nomTournois = trame.section(";",4,4);
00347         darts->configurationTournois(joueurs, modeJeu, nomTournois);
00348
```

```
00349     }
00350     else if(trame.contains("PLAY") && (etatPartie == EtatPartie::AttenteTournois))
00351     {
00352         darts->demarrerTournois();
00353     }
00354     else
00355     {
00356         qDebug() << Q_FUNC_INFO << "Trame non Traité: " << trame;
00357     }
00358 }
```

7.1.4.7 demarrer()

```
void Communication::demarrer ( )
```

Méthode qui démarre le serveur.

Démarre le serveur.

Définition à la ligne 103 du fichier communication.cpp.

Références localDevice, nouveauClient(), serveur, serviceInfo, serviceNom(), et serviceUuid().

Référencé par Ihm : :Ihm().

```
00104 {
00105     if (!localDevice.isValid())
00106         return;
00107
00108     if (!serveur)   //Démarre le serveur s'il n'est pas déjà démarré
00109     {
00110         serveur = new QBluetoothServer(QBluetoothServiceInfo::RfcommProtocol, this);
00111         connect(serveur, SIGNAL(newConnection()), this, SLOT(
      nouveauClient()));
00112
00113         QBluetoothUuid uuid = QBluetoothUuid(serviceUuid);
00114         serviceInfo = serveur->listen(uuid, serviceNom);
00115     }
00116 }
```

7.1.4.8 deviceConnected

```
void Communication::deviceConnected (
             const QBluetoothAddress & adresse )  [private], [slot]
```

Slot appelé quand un nouvel appareil est connecté

Méthode appelée quand l'appareil est connecté

Paramètres

adresse	

Définition à la ligne 415 du fichier communication.cpp.

Références appareilConnecter(), etatPartie, localDevice, play(), et relancerPartie().

Référencé par parametrerBluetooth().

```
00416 {
00417     qDebug() << Q_FUNC_INFO << adresse << localDevice.pairingStatus(adresse);
00418     QString message = QString::fromUtf8("Demande connexion du client ") + adresse.toString();
00419     emit appareilConnecter();
00420     if (localDevice.pairingStatus(adresse) == QBluetoothLocalDevice::Paired ||
      localDevice.pairingStatus(adresse) == QBluetoothLocalDevice::AuthorizedPaired)
00421         message += " [" + QString::fromUtf8("appairé") + "]";
00422     else
00423         message += " [" + QString::fromUtf8("non appairé") + "]" ;
00424     qDebug() << message << endl;
00425
00426     if(etatPartie == EtatPartie::Pause) // si l'appareil est reconnecte, la partie reprend
00427     {
00428         emit play();
00429         relancerPartie();
00430     }
00431 }
```

7.1.4.9 deviceDisconnected

```
void Communication::deviceDisconnected (
            const QBluetoothAddress & adresse )  [private], [slot]
```

Slot appelé quand un appareil est déconnecté

Méthode appelée quand l'appareil est deconnecté

Paramètres

adresse	

Définition à la ligne 439 du fichier communication.cpp.

Références afficherAttenteConnexion(), etatPartie, etatPrecedent, miseAJourEtatPartiePause(), et pause().

Référencé par parametrerBluetooth().

```
00440 {
00441     qDebug() << Q_FUNC_INFO << adresse;
00442     emit afficherAttenteConnexion();
00443
00444     if(etatPartie == EtatPartie::EnCours || etatPartie == EtatPartie::Tournois) // si
       l'appareil se deconnecte pendant la partie, il la met donc en pause
00445     {
00446         etatPrecedent = etatPartie;
00447         emit pause();
00448         miseAJourEtatPartiePause();
00449     }
00450 }
```

7.1.4.10 erreurBluetooth

```
void Communication::erreurBluetooth (
            QString erreur )  [signal]
```

signal emit quand un probleme de configuration bluetooth est detecté

Référencé par parametrerBluetooth().

7.1.4.11 error

```
void Communication::error (
            QBluetoothLocalDevice::Error erreur )  [private], [slot]
```

Slot appelé quand il y a une erreur avec l'appareil bluetooth.

Méthode appelée quand il y a une erreur avec l'appareil connecté

Paramètres

erreur	

Définition à la ligne 458 du fichier communication.cpp.

Référencé par parametrerBluetooth().

```
00459 {
00460     qDebug() << Q_FUNC_INFO << erreur;
00461 }
```

7.1.4.12 estValide()

```
bool Communication::estValide ( )  [private]
```

Méthode qui test si la trame est valide.

Méthode qui teste si la trame reçu est valide.

Renvoie

bool

Définition à la ligne 261 du fichier communication.cpp.

Références DELIMITEUR_FIN, trame, et TYPE_TRAME.

Référencé par decomposerTrame().

```
00262 {
00263     if(trame.startsWith(TYPE_TRAME) && trame.endsWith(
      DELIMITEUR_FIN) && trame.contains(";"))
00264     {
00265         return true;
00266     }
00267     else
00268     {
00269         qDebug() << Q_FUNC_INFO << "Trame non Valide: " << trame;
00270         return false;
00271     }
00272 }
```

7.1.4.13 extraireParametresTrameRegle()

```
void Communication::extraireParametresTrameRegle ( )  [private]
```

Méthode qui extrait les paramètres de la trame REGLE.

Méthode qui decompose la trame regle.

Définition à la ligne 365 du fichier communication.cpp.

Références afficherRegle(), darts, etatPartie, pause(), Darts : :testerModeDeJeu(), et trame.

Référencé par decomposerTrame().

```
00366 {
00367     QString regle ="";
00368
00369     if(etatPartie != EtatPartie::Pause)
00370         emit pause();
00371
00372     if(trame.section(";",2,2).contains("DOUBLE_OUT"))
00373     {
00374         emit afficherRegle("DOUBLE_OUT");
00375     }
00376     else if(trame.section(";",2,2) == "" && (etatPartie == EtatPartie::EnCours ||
      etatPartie == EtatPartie::Pause))
00377     {
00378         emit afficherRegle(darts->testerModeDeJeu());
00379     }
00380     else
00381     {
00382         emit afficherRegle("SANS_DOUBLE_OUT");
00383     }
00384 }
```

7.1.4.14 extraireParametresTrameStart()

```
void Communication::extraireParametresTrameStart (
            QStringList & joueurs,
            QString & modeJeu )  [private]
```

Méthode qui extrait les paramètres de la trame START.

Méthode qui decompose la trame Start.

Paramètres

joueurs	
modeJeu	

Définition à la ligne 281 du fichier communication.cpp.

Références afficherRegle(), darts, etatPartie, Darts : :initialiserPartie(), pause(), Darts : :testerModeDeJeu(), et trame.

Référencé par decomposerTrame().

```
00282 {
00283     modeJeu = trame.section(";",2,2);
00284
00285     if(trame.section(";",4,4).toInt() == 0) //test effectuer pour verifier que la trame n'est pas une
       trame de la version du protocol DARTS 0.2
00286         return;
```

```
00287
00288     for(int i = 0;i <= trame.section(";",4,4).toInt();i++)  //boucle qui recuperer les noms des
      differents joueurs
00289     {
00290         if(trame.section(";",4+i,4+i) == "")    //test si le joueur a un nom
00291         {
00292             joueurs.push_back("Joueur[" + QString::number(i) + "]");
00293         }
00294         else
00295         {
00296             joueurs.push_back(trame.section(";",4+i,4+i));
00297         }
00298     }
00299
00300     darts->initialiserPartie(joueurs, modeJeu);
00301
00302     if(trame.section(";",3,3) == "1")
00303     {
00304         if(etatPartie != EtatPartie::Pause)
00305             emit pause();
00306
00307         emit afficherRegle(darts->testerModeDeJeu());
00308     }
00309
00310 }
```

7.1.4.15 getEtatPartie()

```
int Communication::getEtatPartie ( )
```

Méthode qui permet de recuperer l'etat de la partie.

Méthode qui retourne l'état de la partie.

Renvoie

int

Définition à la ligne 49 du fichier communication.cpp.

Références etatPartie.

Référencé par Ihm : :lancerRegle().

```
00050 {
00051     return etatPartie;
00052 }
```

7.1.4.16 jouerSon

```
void Communication::jouerSon (
             QString  )  [signal]
```

Référencé par decomposerTrame().

7.1.4.17 miseAJourEtatPartieAttente()

```
void Communication::miseAJourEtatPartieAttente ( )
```

Méthode appelé pour mettre à jour l'état de la partie à Attente.

Méthode qui met à jour l'état de la partie Attente.

Définition à la ligne 468 du fichier communication.cpp.

Références etatPartie.

Référencé par Communication(), reamorcerPartie(), et Ihm : :testerEtatPartie().

```
00469 {
00470     qDebug() << Q_FUNC_INFO << "EtatPartie::Attente";
00471     etatPartie = EtatPartie::Attente;
00472 }
```

7.1.4.18 miseAJourEtatPartieAttenteTournois

```
void Communication::miseAJourEtatPartieAttenteTournois ( )  [slot]
```

Slot appelé pour mettre à jour l'état de la partie à attenteTournois.

Méthode qui met à jour l'état de la partie en mode tournois.

Définition à la ligne 535 du fichier communication.cpp.

Références etatPartie.

Référencé par parametrerBluetooth().

```
00536 {
00537     qDebug() << Q_FUNC_INFO << "EtatPartie::AttenteTournois";
00538     etatPartie = EtatPartie::AttenteTournois;
00539 }
```

7.1.4.19 miseAJourEtatPartieEnCours

```
void Communication::miseAJourEtatPartieEnCours ( )  [slot]
```

Slot appelé pour mettre à jour l'état de la partie à EnCours.

Méthode qui met à jour l'état de la partie EnCours.

Définition à la ligne 501 du fichier communication.cpp.

Références etatPartie.

Référencé par parametrerBluetooth(), relancerPartie(), et Ihm : :testerEtatPartie().

```
00502 {
00503     qDebug() << Q_FUNC_INFO << "EtatPartie::EnCours";
00504     etatPartie = EtatPartie::EnCours;
00505 }
```

7.1.4.20 miseAJourEtatPartieFin

```
void Communication::miseAJourEtatPartieFin ( )  [slot]
```

Slot appelé pour mettre à jour l'état de la partie à Fin.

Méthode qui met à jour l'état de la partie Fin.

Définition à la ligne 490 du fichier communication.cpp.

Références etatPartie.

Référencé par parametrerBluetooth(), et Ihm : :testerEtatPartie().

```
00491 {
00492     qDebug() << Q_FUNC_INFO << "EtatPartie::Fin";
00493     etatPartie = EtatPartie::Fin;
00494 }
```

7.1.4.21 miseAJourEtatPartiePause()

```
void Communication::miseAJourEtatPartiePause ( )
```

Méthode appelé pour mettre à jour l'état de la partie à Pause.

Méthode qui met à jour l'état de la partie Pause.

Définition à la ligne 479 du fichier communication.cpp.

Références etatPartie.

Référencé par decomposerTrame(), deviceDisconnected(), et Ihm : :testerEtatPartie().

```
00480 {
00481     qDebug() << Q_FUNC_INFO << "EtatPartie::Pause";
00482     etatPartie = EtatPartie::Pause;
00483 }
```

7.1.4.22 miseAJourEtatPartieRegle()

```
Communication::miseAJourEtatPartieRegle ( )
```

Méthode appelé pour mettre à jour l'état de la partie à regle.

Méthode qui met à jour l'état de la partie en mode tournois.

Méthode qui met à jour l'état de la partie en regle.

Définition à la ligne 512 du fichier communication.cpp.

Références etatPartie.

Référencé par Ihm : :lancerRegle().

```
00513 {
00514     qDebug() << Q_FUNC_INFO << "EtatPartie::Regle";
00515     etatPartie = EtatPartie::Regle;
00516 }
```

7.1.4.23 miseAJourEtatPartieTournois

```
void Communication::miseAJourEtatPartieTournois ( )  [slot]
```

Slot appelé pour mettre à jour l'état de la partie à tournois.

Définition à la ligne 523 du fichier communication.cpp.

Références etatPartie.

Référencé par parametrerBluetooth(), et relancerPartie().

```
00524 {
00525     qDebug() << Q_FUNC_INFO << "EtatPartie::Tournois";
00526     etatPartie = EtatPartie::Tournois;
00527 }
```

7.1.4.24 nouveauClient

```
void Communication::nouveauClient ( )  [private], [slot]
```

Slot appelé quand un nouveau client veut se connecter.

Méthode appelée quand un nouveau client se connecte.

Définition à la ligne 152 du fichier communication.cpp.

Références serveur, socket, socketDisconnected(), et socketReadyRead().

Référencé par demarrer().

```
00153 {
00154     // on récupère la socket
00155     socket = serveur->nextPendingConnection();
00156     if (!socket)
00157         return;
00158
00159     connect(socket, SIGNAL(disconnected()), this, SLOT(socketDisconnected()));
00160     connect(socket, SIGNAL(readyRead()), this, SLOT(socketReadyRead()));
00161 }
```

7.1.4.25 parametrerBluetooth()

```
void Communication::parametrerBluetooth ( )
```

Méthode qui configure la connexion Bluetooth en mode serveur.

configure la communication Bluetooth

Définition à la ligne 59 du fichier communication.cpp.

Références darts, deviceConnected(), deviceDisconnected(), erreurBluetooth(), error(), localDevice, localDeviceName, miseAJour↩EtatPartieAttenteTournois(), miseAJourEtatPartieEnCours(), miseAJourEtatPartieFin(), et miseAJourEtatPartieTournois().

Référencé par Communication().

```
00060 {
00061     if (!localDevice.isValid())
00062     {
00063         qDebug() << Q_FUNC_INFO << "Communication Bluetooth locale valide : " <<
      localDevice.isValid();
00064         emit erreurBluetooth("Communication Bluetooth locale valide : " + QString::number(
      localDevice.isValid()));
00065
00066         return;
00067     }
00068     else
00069     {
00070         // active le bluetooth
00071         localDevice.powerOn();
00072
00073         // lire le nom de l'appareil local
00074         localDeviceName = localDevice.name();
00075
00076
00077         //localDevice.setHostMode(QBluetoothLocalDevice::HostConnectable);
00078
00079         //ou
00080
00081         //les appareil qui ne sont pas jumelé peuvent decouvrir ecran-DARTS
00082         localDevice.setHostMode(QBluetoothLocalDevice::HostDiscoverable);
00083
00084         QList<QBluetoothAddress> remotes;
00085         remotes = localDevice.connectedDevices();
00086
00087         connect(&localDevice, SIGNAL(deviceConnected(QBluetoothAddress)), this,
      SLOT(deviceConnected(QBluetoothAddress)));
00088         connect(&localDevice, SIGNAL(deviceDisconnected(QBluetoothAddress)),
      this, SLOT(deviceDisconnected(QBluetoothAddress)));
00089         connect(&localDevice, SIGNAL(error(QBluetoothLocalDevice::Error)), this, SLOT(
      error(QBluetoothLocalDevice::Error)));
00090
00091         connect(darts, SIGNAL(etatPartieFini()), this , SLOT(
      miseAJourEtatPartieFin()));
00092         connect(darts, SIGNAL(etatPartieTournois()), this , SLOT(
      miseAJourEtatPartieTournois()));
00093         connect(darts, SIGNAL(changerEtatPartie()), this , SLOT(
      miseAJourEtatPartieEnCours()));
00094         connect(darts, SIGNAL(etatPartieAttenteTournois()), this, SLOT(
      miseAJourEtatPartieAttenteTournois()));
00095     }
00096 }
```

7.1.4.26 pause

```
void Communication::pause ( )  [signal]
```

signal qui mettra en pause la partie

Référencé par decomposerTrame(), deviceDisconnected(), extraireParametresTrameRegle(), et extraireParametresTrameStart().

7.1.4.27 play

```
void Communication::play ( )  [signal]
```

signal qui relancera le chronometrage de la partie la partie

Référencé par decomposerTrame(), et deviceConnected().

7.1.4.28 reamorcerPartie()

```
void Communication::reamorcerPartie ( )  [private]
```

Méthode qui relance la partie.

Méthode qui reinitialise la partie.

Définition à la ligne 391 du fichier communication.cpp.

Références darts, miseAJourEtatPartieAttente(), Darts : :reinitialiserPartie(), et resetPartie().

Référencé par decomposerTrame().

```
00392 {
00393     emit resetPartie();
00394     darts->reinitialiserPartie();
00395     miseAJourEtatPartieAttente();
00396 }
```

7.1.4.29 relancerPartie()

```
void Communication::relancerPartie ( )  [private]
```

Méthode qui gere l'initialisation d'une partie de flechette.

Methode qui remet la partie dans l'etat dans lequel elle se trouver avant la pause.

Définition à la ligne 546 du fichier communication.cpp.

Références etatPrecedent, miseAJourEtatPartieEnCours(), et miseAJourEtatPartieTournois().

Référencé par decomposerTrame(), et deviceConnected().

```
00547 {
00548     if(etatPrecedent == EtatPartie::EnCours)
00549     {
00550         miseAJourEtatPartieEnCours();
00551     }
00552     else
00553     {
00554         miseAJourEtatPartieTournois();
00555     }
00556 }
```

7.1.4.30 resetPartie

```
void Communication::resetPartie ( )  [signal]
```

signal qui reinitialisera la partie en cours

Référencé par decomposerTrame(), et reamorcerPartie().

7.1.4.31 socketDisconnected

```
void Communication::socketDisconnected ( )  [private], [slot]
```

Slot appelé quand la communication est deconnectée.

Méthode appelée quand le socket est déconnecté

Définition à la ligne 403 du fichier communication.cpp.

Référencé par nouveauClient().

```
00404 {
00405     QString message = QString::fromUtf8("Périphérique déconnecté ");
00406     qDebug() << Q_FUNC_INFO << message;
00407 }
```

7.1.4.32 socketReadyRead

```
void Communication::socketReadyRead ( )  [private], [slot]
```

Slot appelé quand une nouvelle trame est disponible.

Méthode appelée quand une trame est disponible.

Définition à la ligne 168 du fichier communication.cpp.

Références decomposerTrame(), socket, et trame.

Référencé par nouveauClient().

```
00169 {
00170     QByteArray donnees;
00171
00172     while (socket->bytesAvailable())
00173     {
00174         donnees += socket->readAll();
00175         usleep(150000); // cf. timeout
00176     }
00177
00178     trame = QString(donnees);
00179
00180     qDebug() << Q_FUNC_INFO << QString::fromUtf8("Trame reçues : ") << QString(donnees);
00181
00182     decomposerTrame();
00183 }
```

7.1.4.33 stopperRegle

```
void Communication::stopperRegle ( )  [signal]
```

signal emit pour stopper la lecture des regles

Référencé par decomposerTrame().

7.1.5 Documentation des données membres

7.1.5.1 darts

```
Darts* Communication::darts  [private]
```

Association avec l'objet darts.

Définition à la ligne 99 du fichier communication.h.

Référencé par decomposerTrame(), decomposerTrameTournois(), extraireParametresTrameRegle(), extraireParametresTrameStart(), parametrerBluetooth(), et reamorcerPartie().

7.1.5.2 etatPartie

```
EtatPartie Communication::etatPartie  [private]
```

L'état de la partie.

Définition à la ligne 106 du fichier communication.h.

Référencé par decomposerTrame(), decomposerTrameTournois(), deviceConnected(), deviceDisconnected(), extraireParametres↩TrameRegle(), extraireParametresTrameStart(), getEtatPartie(), miseAJourEtatPartieAttente(), miseAJourEtatPartieAttenteTournois(), miseAJourEtatPartieEnCours(), miseAJourEtatPartieFin(), miseAJourEtatPartiePause(), miseAJourEtatPartieRegle(), et miseAJour↩EtatPartieTournois().

7.1.5.3 etatPrecedent

```
int Communication::etatPrecedent  [private]
```

Contient l'etat dans lequel se trouver l'application.

Définition à la ligne 114 du fichier communication.h.

Référencé par decomposerTrame(), deviceDisconnected(), et relancerPartie().

7.1.5.4 localDevice

```
QBluetoothLocalDevice Communication::localDevice  [private]
```

L'interface Bluetooth de la Raspberry Pi.

Définition à la ligne 102 du fichier communication.h.

Référencé par arreter(), demarrer(), deviceConnected(), parametrerBluetooth(), et ~Communication().

7.1.5.5 localDeviceName

```
QString Communication::localDeviceName  [private]
```

Nom du peripherique local.

Définition à la ligne 104 du fichier communication.h.

Référencé par parametrerBluetooth().

7.1.5.6 serveur

```
QBluetoothServer* Communication::serveur  [private]
```

Le serveur Bluetooth.

Définition à la ligne 100 du fichier communication.h.

Référencé par arreter(), demarrer(), et nouveauClient().

7.1.5.7 serviceInfo

```
QBluetoothServiceInfo Communication::serviceInfo  [private]
```

Informations sur le service bluetooth.

Définition à la ligne 103 du fichier communication.h.

Référencé par arreter(), et demarrer().

7.1.5.8 socket

```
QBluetoothSocket* Communication::socket  [private]
```

La socket de communication Bluetooth.

Définition à la ligne 101 du fichier communication.h.

Référencé par arreter(), nouveauClient(), et socketReadyRead().

7.1.5.9 trame

```
QString Communication::trame  [private]
```

La trame recue.

Définition à la ligne 105 du fichier communication.h.

Référencé par decomposerTrame(), decomposerTrameTournois(), estValide(), extraireParametresTrameRegle(), extraireParametres↩TrameStart(), et socketReadyRead().

La documentation de cette classe a été générée à partir des fichiers suivants :

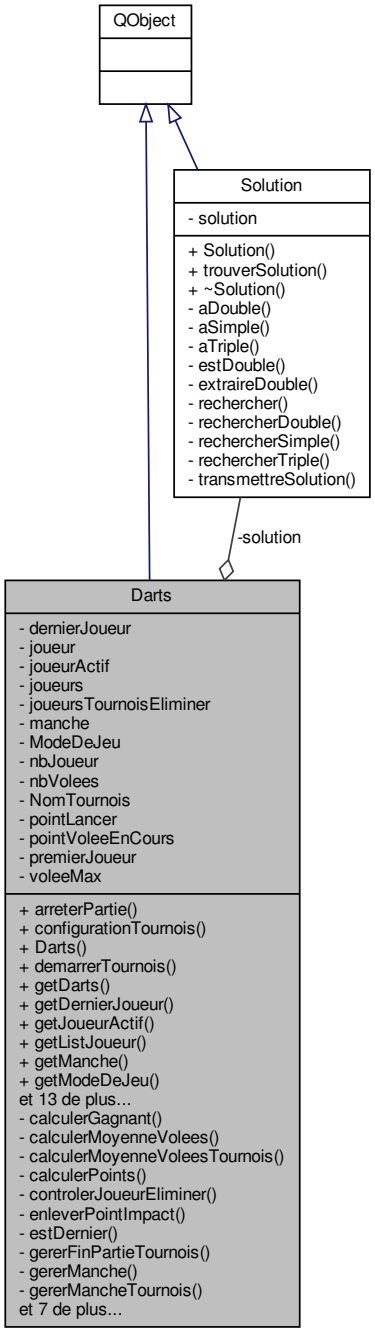
- communication.h
- communication.cpp

7.2 Référence de la classe Darts

Déclaration de la classe Darts (Module Ecran-DARTS)

```
#include <darts.h>
```

Graphe de collaboration de Darts :



Signaux

- void actualiserCible ()
  signal émis pour changer actualiser l'affichage de la cible
- void afficherInfoTournois ()
- void afficherNouvellePartie ()
  signal émis quand il y a une nouvelle partie
- void afficherTournois (QString modeJeu, QString nomTournois)
  signal émis pour afficher le tournois
- void changementJoueurActif ()
  signal émis quand le joueur actif change

- void changementJoueurActifTournoi ()
- void changerEtatPartie ()
  signal émis pour changer l'etat de la partie
- void debuterTournois ()
  signal émis pour debuter le tournois
- void etatPartieAttenteTournois ()
- void etatPartieFini ()
  signal émis pour mettre l'etat de la partie en fin
- void etatPartieTournois ()
  signal émis pour mettre en etat de tournois
- void finPartie (QString, int, bool)
  signal émis quand c'est la fin de la partie
- void finTournois (QString, QString, QList< Joueur >)
- void jouerSon (QString son)
  signal émis pour Lancer un son
- void miseAJourMoyenneVolee ()
  signal émis pour mettre à jour la moyenne des volées
- void miseAJourMoyenneVoleeTournois ()
  signal émis pour mettre a jour la moyenne des Volées en mode tournois
- void miseAJourPoint ()
  signal émis pour mettre à jour les points des joueurs
- void miseAJourPointTournois ()
  signal émis pour mettre a jour les points du tournois
- void nouvelImpact (int, int, int)
  signal émis quand il y a un nouvel Impact
- void nouvelleManche ()
  signal émis quand on change de manche
- void voleeAnnulee ()
  signal émis quand la volées est annulé

Fonctions membres publiques

- void arreterPartie ()
  arrête la partie
- void configurationTournois (QStringList joueurList, QString modeJeu, QString nomTournois)
  methode utiliser pour configurer le tournois
- Darts (QObject *parent=nullptr)
  Constructeur de la classe Darts.
- void demarrerTournois ()
  methode appelé pour demarrer le tournois
- Darts * getDarts () const
- int getDernierJoueur () const
  Retourne le dernier joueur a jouer.
- int getJoueurActif () const
  Retourne le numéro du joueur actif.
- QList< Joueur > getListJoueur () const
  Retourne une liste des joueurs.
- int getManche () const
  Retourne la manche.
- QString getModeDeJeu () const
  Retourne le mode de jeu.
- int getNbVolees () const
  Retourne le nombre de volées de la partie.
- int getPointVolees () const
  Retourne la manche.
- int getPremierJoueur () const
  Retourne le premier Joueur a jouer.
- Solution * getSolution () const
  retourne l'objet solution
- int getVoleeMax () const
  Retourne la volée max.
- void initialiserPartie (QStringList joueurList, QString modeJeu)
  Initialise la partie.
- void receptionnerImpact (int typePoint, int point)
  Permets de traiter la réception d'impact.
- void receptionnerImpactTournois (int typePoint, int point)
  methode qui gere le deroulement de la partie tournois
- void reinitialiserPartie ()
  Méthode qui permet de remettre a zero les differents attribut et conteneur pour une nouvelle partie.
- void setManche (int manche)
  Permets de mettre à jour le numéro de manche.

- void setVoleeMax (int voleeMax)
  Permets de mettre à jour le volé max.
- QString testerModeDeJeu ()
  Méthode qui teste le mode de jeu.
- ~Darts ()
  Destructeur de la classe Darts.

Fonctions membres privées

- QString calculerGagnant ()
  Calcule le gagnant de la partie si la partie doit s'arrêter avant la fin.
- void calculerMoyenneVolees ()
  Calcule la moyenne des volées de chaque joueur.
- void calculerMoyenneVoleesTournois ()
  Calcule la moyenne des volées de chaque joueur pour le tournois.
- void calculerPoints (int point, int typePoint)
  Teste s'il reste qu'un joueur n'était pas éliminé
- void controlerJoueurEliminer ()
  Change de joueur si le joueur actuel est éliminé
- void enleverPointImpact ()
  Met à jours le score du joueur.
- bool estDernier ()
  test pour savoir si la personne a gagner le tournois
- void gererFinPartieTournois ()
  methode qui gere le deroulement en fin de manche
- void gererManche ()
  Permets de gérer le changement de manche en fonction des fléchettes de chaque joueur.
- void gererMancheTournois ()
  Permets de gérer le changement de manche en fonction des fléchettes de chaque joueur.
- void gererVoleeMax ()
  Teste la volée pour savoir si elle est supérieure à la volée Max.
- void initialiserFinTournois ()
  methode qui gere la fin du tournois
- void testerImpact (int typePoint)
  Teste si le joueur a gagné
- void testerImpactTournois (int typePoint)
  teste l'impact pour le mode tournois
- void testerNombreJoueurRestand ()
  Teste s'il reste qu'un joueur n'était pas éliminé
- void testerPoint (int typePoint, int point)
  Méthode qui teste les Impact pour savoir quel son jouer.
- void testerSiJoueurEliminer ()
  Teste si le joueur est à 1 point à la fin de la manche.

Attributs privés

- int dernierJoueur
  contient le dernier joueur du tournois a jouer
- QStringList joueur
  contient les noms des differents joueur
- int joueurActif
  contient le numero du joueur en train de jouer
- QList< Joueur > joueurs
  contient des objets joueurs
- QList< Joueur > joueursTournoisEliminer
  contient des objets joueurs eliminer pendant le tournois
- int manche
  contient le numero de la manche actuel
- QString ModeDeJeu
  contient le mode de jeu en cours
- int nbJoueur
  contient le nombre de joueur
- int nbVolees
  contient le nombre de Volées de la partie en cours
- QString NomTournois
  contient le nom du tournois
- int pointLancer
  contient les point associer l'impact de la fleche

— int pointVoleeEnCours
contient le score de la Volées en cours
— int premierJoueur
contient le premier joueur du tournois a jouer
— Solution * solution
Association vers l'objet solution.
— int voleeMax
contient la volées Max

7.2.1 Description détaillée

Déclaration de la classe Darts (Module Ecran-DARTS)

Cette classe s'occuper du déroulement d'une partie

Définition à la ligne 33 du fichier darts.h.

7.2.2 Documentation des constructeurs et destructeur

7.2.2.1 Darts()

```
Darts::Darts (
            QObject * parent = nullptr )  [explicit]
```

Constructeur de la classe Darts.

Paramètres

parent	

Définition à la ligne 22 du fichier darts.cpp.

Références solution.

```
00022                           : QObject(parent), joueur(nullptr),
      nbJoueur(0), joueurActif(0), manche(1), pointLancer(0),
      voleeMax(0), nbVolees(0), ModeDeJeu(""),
      pointVoleeEnCours(0)
00023 {
00024     qDebug() << Q_FUNC_INFO;
00025     solution = new Solution(this);
00026 }
```

7.2.2.2 ~Darts()

```
Darts::~Darts ( )
```

Destructeur de la classe Darts.

Définition à la ligne 33 du fichier darts.cpp.

```
00034 {
00035     qDebug() << Q_FUNC_INFO;
00036 }
```

7.2.3 Documentation des fonctions membres

7.2.3.1 actualiserCible

```
void Darts::actualiserCible ( )  [signal]
```

signal émis pour changer actualiser l'affichage de la cible

Référencé par demarrerTournois(), receptionnerImpact(), et receptionnerImpactTournois().

7.2.3.2 afficherInfoTournois

```
void Darts::afficherInfoTournois ( )  [signal]
```

Référencé par configurationTournois(), et testerImpactTournois().

7.2.3.3 afficherNouvellePartie

```
void Darts::afficherNouvellePartie ( )  [signal]
```

signal émis quand il y a une nouvelle partie

Référencé par initialiserPartie().

7.2.3.4 afficherTournois

```
void Darts::afficherTournois (
            QString modeJeu,
            QString nomTournois )  [signal]
```

signal émis pour afficher le tournois

Référencé par configurationTournois(), et gererFinPartieTournois().

7.2.3.5 arreterPartie()

```
void Darts::arreterPartie ( )
```

arrête la partie

Définition à la ligne 462 du fichier darts.cpp.

Références calculerGagnant(), etatPartieFini(), finPartie(), et getVoleeMax().

Référencé par Communication : :decomposerTrame(), et testerNombreJoueurRestand().

```
00463 {
00464     emit finPartie(calculerGagnant(), getVoleeMax(), false);
00465     emit etatPartieFini();
00466 }
```

7.2.3.6 calculerGagnant()

```
QString Darts::calculerGagnant ( )  [private]
```

Calcule le gagnant de la partie si la partie doit s'arrêter avant la fin.

Renvoie

QString

Définition à la ligne 474 du fichier darts.cpp.

Références joueurs.

Référencé par arreterPartie().

```
00475 {
00476     QString gagnant;
00477     int scoreGagnant = 99999;
00478     for(int i = 0; i <= joueurs.size() - 1; i++)
00479     {
00480         if(scoreGagnant > joueurs[i].getScore() && !joueurs[i].getEliminer()) // test le
      personne aillant le score le plus bas score mais aussi qu'il ne soit pas eliminer
00481         {
00482             scoreGagnant = joueurs[i].getScore();
00483             gagnant = joueurs[i].getNom();
00484         }
00485     }
00486     return "  Winner " + gagnant + "  ";
00487 }
```

7.2.3.7 calculerMoyenneVolees()

```
void Darts::calculerMoyenneVolees ( )  [private]
```

Calcule la moyenne des volées de chaque joueur.

Définition à la ligne 427 du fichier darts.cpp.

Références joueurs, et miseAJourMoyenneVolee().

Référencé par controlerJoueurEliminer(), et gererManche().

```
00428 {
00429     for(int i = 0; i <= joueurs.size() - 1; i++)
00430     {
00431         int moyenneVolee = 0;
00432
00433         for(int j = 0; j < joueurs[i].getHistoriqueVolees().size(); j++)
00434         {
00435             moyenneVolee += joueurs[i].getHistoriqueVolees()[j];
00436         }
00437
00438         moyenneVolee = moyenneVolee / joueurs[i].getHistoriqueVolees().size();
00439         joueurs[i].setMoyenneVolee(moyenneVolee);
00440     }
00441     emit miseAJourMoyenneVolee();
00442 }
```

7.2.3.8 calculerMoyenneVoleesTournois()

```
void Darts::calculerMoyenneVoleesTournois ( )  [private]
```

Calcule la moyenne des volées de chaque joueur pour le tournois.

Définition à la ligne 730 du fichier darts.cpp.

Références dernierJoueur, joueurs, miseAJourMoyenneVoleeTournois(), et premierJoueur.

Référencé par gererMancheTournois().

```
00731 {
00732     for(int i = premierJoueur; i <= dernierJoueur; i++)
00733     {
00734         int moyenneVolee = 0;
00735
00736         for(int j = 0; j < joueurs[i].getHistoriqueVolees().size(); j++)
00737         {
00738             moyenneVolee += joueurs[i].getHistoriqueVolees()[j];
00739         }
00740
00741         moyenneVolee = moyenneVolee / joueurs[i].getHistoriqueVolees().size();
00742         joueurs[i].setMoyenneVolee(moyenneVolee);
00743     }
00744     emit miseAJourMoyenneVoleeTournois();
00745 }
```

7.2.3.9 calculerPoints()

```
void Darts::calculerPoints (
            int point,
            int typePoint )  [private]
```

Teste s'il reste qu'un joueur n'était pas éliminé

Paramètres

point	la zone
typePoint	un simple, double ou triple

Définition à la ligne 519 du fichier darts.cpp.

Références DOUBLE_POINT, pointLancer, SIMPLE_POINT, TRIPLE_POINT, et ZERO_POINT.

Référencé par receptionnerImpact(), et receptionnerImpactTournois().

```
00520 {
00521     switch(typePoint)
00522     {
00523         case TRIPLE_POINT:
00524             pointLancer = point * TRIPLE_POINT;
00525         break;
00526         case DOUBLE_POINT:
00527             pointLancer = point * DOUBLE_POINT;
00528         break;
00529         case SIMPLE_POINT:
00530             pointLancer = point;
00531         break;
00532         case  ZERO_POINT:
00533             pointLancer = point * ZERO_POINT;
00534         break;
00535         default:
00536             qDebug() << Q_FUNC_INFO << "type de point non valide:" << typePoint;
00537         break;
00538     }
00539 }
```

7.2.3.10 changementJoueurActif

```
void Darts::changementJoueurActif ( )  [signal]
```

signal émis quand le joueur actif change

Référencé par gererManche().

7.2.3.11 changementJoueurActifTournoi

```
void Darts::changementJoueurActifTournoi ( )  [signal]
```

Référencé par demarrerTournois(), et gererMancheTournois().

7.2.3.12 changerEtatPartie

```
void Darts::changerEtatPartie ( )  [signal]
```

signal émis pour changer l'etat de la partie

Référencé par initialiserPartie().

7.2.3.13 configurationTournois()

```
void Darts::configurationTournois (
            QStringList joueurList,
            QString modeJeu,
            QString nomTournois )
```

methode utiliser pour configurer le tournois

Paramètres

joueurList	
modeJeu	
nomTournois	

Définition à la ligne 570 du fichier darts.cpp.

Références afficherInfoTournois(), afficherTournois(), dernierJoueur, etatPartieAttenteTournois(), joueurActif, joueurs, ModeDeJeu, NB_FLECHETTE, nbJoueur, NomTournois, premierJoueur, reinitialiserPartie(), solution, et Solution : :trouverSolution().

Référencé par Communication : :decomposerTrameTournois().

```
00571 {
00572     nbJoueur = joueurList.size() - 1;
00573     ModeDeJeu = modeJeu;
```

```
00574     NomTournois = nomTournois;
00575     qDebug() << Q_FUNC_INFO << "Nom Tournois " << nomTournois << "nbJoueur " <<
      nbJoueur << " | Mode De Jeu " << ModeDeJeu;
00576
00577     if(ModeDeJeu.toInt() >= 101 && ModeDeJeu.toInt() <= 1001)
00578     {
00579         for(int i = 1; i < joueurList.size() ; i++)
00580         {
00581             Joueur player(joueurList.at(i), ModeDeJeu.toInt(),
      NB_FLECHETTE);
00582             joueurs.push_back(player);
00583         }
00584     }
00585     else if(ModeDeJeu.contains("_DOUBLE_OUT") && (ModeDeJeu.section("_",0,0).toInt() >= 101 && ModeDeJeu.
      section("_",0,0).toInt() <= 1001))
00586     {
00587         for(int i = 1; i < joueurList.size() ; i++)
00588         {
00589             Joueur player(joueurList.at(i), ModeDeJeu.section("_",0,0).toInt(),
      NB_FLECHETTE);
00590             joueurs.push_back(player);
00591         }
00592     }
00593     else
00594     {
00595         qDebug() << Q_FUNC_INFO << "Erreur Mode De Jeu : " << ModeDeJeu;
00596         reinitialiserPartie();
00597         return;
00598     }
00599     solution->trouverSolution(joueurs[joueurActif].getScore(),
      joueurs[joueurActif].getFlechette());
00600
00601     premierJoueur = 0;
00602     dernierJoueur = 1;
00603
00604     emit afficherTournois(modeJeu, nomTournois);
00605     emit etatPartieAttenteTournois();
00606     emit afficherInfoTournois();
00607 }
```

7.2.3.14 controlerJoueurEliminer()

```
void Darts::controlerJoueurEliminer ( )  [private]
```

Change de joueur si le joueur actuel est éliminé

Définition à la ligne 391 du fichier darts.cpp.

Références calculerMoyenneVolees(), getManche(), joueurActif, joueurs, nbJoueur, nouvelleManche(), et setManche().

Référencé par gererManche().

```
00392 {
00393     while(joueurs[joueurActif].getEliminer()) //tant que le joueur est eliminer on passe
       donc au joueur suivant
00394     {
00395         if(joueurActif == nbJoueur - 1)  //si le joueur est le dernier de la manche à
       jouer
00396         {
00397             joueurActif = 0;
00398             setManche(getManche() + 1);
00399             calculerMoyenneVolees();
00400             emit nouvelleManche();
00401         }
00402         else                            //si le joueur n'est pas le dernier de la manche
00403         {
00404             joueurActif++;
00405         }
00406     }
00407 }
```

7.2.3.15 debuterTournois

```
void Darts::debuterTournois ( )  [signal]
```

signal émis pour debuter le tournois

Référencé par demarrerTournois().

7.2.3.16 demarrerTournois()

```
void Darts::demarrerTournois ( )
```

methode appelé pour demarrer le tournois

Définition à la ligne 614 du fichier darts.cpp.

Références actualiserCible(), changementJoueurActifTournoi(), debuterTournois(), et etatPartieTournois().

Référencé par Communication : :decomposerTrameTournois().

```
00615 {
00616     emit actualiserCible();
00617     emit changementJoueurActifTournoi();
00618     emit debuterTournois();
00619     emit etatPartieTournois();
00620 }
```

7.2.3.17 enleverPointImpact()

```
void Darts::enleverPointImpact ( )  [private]
```

Met à jours le score du joueur.

Définition à la ligne 314 du fichier darts.cpp.

Références etatPartieFini(), finPartie(), gererVoleeMax(), getVoleeMax(), jouerSon(), joueurActif, joueurs, nbVolees, solution, Solution : :trouverSolution(), et voleeAnnulee().

Référencé par testerImpact(), et testerImpactTournois().

```
00315 {
00316     if(joueurs[joueurActif].getScore() <= 0) //score Volée inferieure au score = Volée
       annulée
00317     {
00318         joueurs[joueurActif].setScore(joueurs[
      joueurActif].getScoreManchePrecedente());
00319         emit voleeAnnulee();
00320
00321         joueurs[joueurActif].setNbFlechette(0);
00322     }
00323     else if(joueurs[joueurActif].getScore() == 1 && joueurs.size() == 1)    //
       test si le joueur est seul à jouer et s'il est a 1 point == joueur éliminer donc fin de partie
00324     {
00325         gererVoleeMax();
00326         emit jouerSon("perdu.wav");
00327         emit finPartie("  " + joueurs[joueurActif].getNom() + " a perdu  ",
      getVoleeMax(), false);
00328         emit etatPartieFini();
00329     }
00330     else
00331     {
00332         nbVolees++;
00333         joueurs[joueurActif].setNbFlechette(joueurs[
      joueurActif].getFlechette() - 1);
00334         solution->trouverSolution(joueurs[
      joueurActif].getScore(),joueurs[joueurActif].getFlechette());
00335     }
00336 }
```

7.2.3.18 estDernier()

```
bool Darts::estDernier ( )  [private]
```

test pour savoir si la personne a gagner le tournois

Renvoie

bool

Définition à la ligne 818 du fichier darts.cpp.

Références joueurs.

Référencé par gererFinPartieTournois().

```
00819 {
00820     int estdernier = 0;
00821     for(int i = 0; i < joueurs.size(); i++)
00822     {
00823         qDebug() << joueurs[i].getNom() << " : " << joueurs[i].getEliminer() << endl;
00824         if(joueurs[i].getEliminer() == false)
00825             estdernier++;
00826     }
00827
00828     if(estdernier == 1)
00829         return true;
00830
00831     return false;
00832 }
```

7.2.3.19 etatPartieAttenteTournois

```
void Darts::etatPartieAttenteTournois ( )  [signal]
```

Référencé par configurationTournois(), et testerImpactTournois().

7.2.3.20 etatPartieFini

```
void Darts::etatPartieFini ( )  [signal]
```

signal émis pour mettre l'etat de la partie en fin

Référencé par arreterPartie(), enleverPointImpact(), gererFinPartieTournois(), et testerImpact().

7.2.3.21 etatPartieTournois

```
void Darts::etatPartieTournois ( )  [signal]
```

signal émis pour mettre en etat de tournois

Référencé par demarrerTournois().

7.2.3.22 finPartie

```
void Darts::finPartie (
            QString ,
            int ,
            bool  )  [signal]
```

signal émis quand c'est la fin de la partie

Référencé par arreterPartie(), enleverPointImpact(), testerImpact(), et testerImpactTournois().

7.2.3.23 finTournois

```
void Darts::finTournois (
            QString ,
            QString ,
            QList< Joueur >  )  [signal]
```

Référencé par initialiserFinTournois().

7.2.3.24 gererFinPartieTournois()

```
void Darts::gererFinPartieTournois ( )  [private]
```

methode qui gere le deroulement en fin de manche

Définition à la ligne 752 du fichier darts.cpp.

Références afficherTournois(), dernierJoueur, estDernier(), etatPartieFini(), initialiserFinTournois(), joueurActif, joueurs, joueursTournoisEliminer, ModeDeJeu, NB_FLECHETTE, nbVolees, NomTournois, premierJoueur, et setManche().

Référencé par testerImpactTournois().

```
00753 {
00754     if(joueurActif == premierJoueur)
00755     {
00756         joueurs[premierJoueur].setEliminer(false);
00757         joueurs[dernierJoueur].setEliminer(true);
00758     }
00759
00760     if(joueurActif == dernierJoueur)
00761     {
00762         joueurs[dernierJoueur].setEliminer(false);
00763         joueurs[premierJoueur].setEliminer(true);
00764     }
00765
00766     if(premierJoueur + 2 < (joueurs.size() - 1)  &&
      dernierJoueur + 2 <= (joueurs.size() - 1))
00767     {
00768         premierJoueur += 2;
00769         dernierJoueur += 2;
00770         nbVolees = 0;
00771
00772         qDebug() << "premierJoueur " << premierJoueur << endl;
00773         qDebug() << "dernierJoueur " << dernierJoueur << endl;
00774     }
00775     else if(estDernier())
00776     {
00777         qDebug() << "Gagnant tournois " << joueurs[joueurActif].getNom() << endl;
00778         initialiserFinTournois();
00779         emit etatPartieFini();
00780     }
00781     else
00782     {
00783         for(int i = joueurs.size() - 1; i >= 0; i--)
00784         {
```

```
00785             if(joueurs[i].getEliminer())
00786             {
00787                 joueursTournoisEliminer.push_back(joueurs[i]);
00788                 joueurs.removeAt(i);
00789             }
00790         }
00791         premierJoueur = 0;
00792         dernierJoueur = 1;
00793         nbVolees = 0;
00794
00795         for(int i = 0; i < joueurs.size(); i++)
00796         {
00797
00798             joueurs[i].setScore(ModeDeJeu.section("_",0,0).toInt());
00799             joueurs[i].setMoyenneVolee(0);
00800             joueurs[i].setScoreManchePrecedente(ModeDeJeu.section("_",0,0).toInt());
00801             joueurs[i].setNbFlechette(NB_FLECHETTE);
00802         }
00803     }
00804
00805     joueurActif = premierJoueur;
00806     setManche(1);
00807
00808     emit afficherTournois(ModeDeJeu, NomTournois);
00809
00810 }
```

7.2.3.25 gererManche()

```
void Darts::gererManche ( )  [private]
```

Permets de gérer le changement de manche en fonction des fléchettes de chaque joueur.

Définition à la ligne 343 du fichier darts.cpp.

Références calculerMoyenneVolees(), changementJoueurActif(), controlerJoueurEliminer(), gererVoleeMax(), getManche(), jouer↩Son(), joueurActif, joueurs, NB_FLECHETTE, nbJoueur, nouvelleManche(), pointVoleeEnCours, setManche(), solution, tester↩NombreJoueurRestand(), testerSiJoueurEliminer(), et Solution : :trouverSolution().

Référencé par testerImpact().

```
00344 {
00345     if(joueurs[joueurActif].getFlechette() == 0) //fin de la volées du joueur
00346     {
00347         joueurs[joueurActif].setNbFlechette(NB_FLECHETTE);
00348
00349         pointVoleeEnCours = 0;
00350
00351         gererVoleeMax();
00352         testerSiJoueurEliminer();
00353         testerNombreJoueurRestand();
00354
00355         joueurs[joueurActif].addHistoriqueVolees((joueurs[
    joueurActif].getScoreManchePrecedente() - joueurs[joueurActif].getScore()));
00356
00357         if((joueurs[joueurActif].getScoreManchePrecedente() -
    joueurs[joueurActif].getScore()) == 180)
00358             emit jouerSon("180.wav");
00359
00360         joueurs[joueurActif].setScoreManchePrecedente(joueurs[
    joueurActif].getScore());
00361
00362         if(joueurActif == nbJoueur - 1)  //equivalent a la fin de manche
00363         {
00364             joueurActif = 0;
00365
00366             controlerJoueurEliminer();
00367
00368             setManche(getManche() + 1);
00369             emit changementJoueurActif();
00370             calculerMoyenneVolees();
00371             emit nouvelleManche();
00372             solution->trouverSolution(joueurs[
    joueurActif].getScore(),joueurs[joueurActif].getFlechette());
00373         }
00374         else
00375         {
```

```
00376             joueurActif++;
00377
00378             controlerJoueurEliminer();
00379
00380             emit changementJoueurActif();
00381             solution->trouverSolution(joueurs[
      joueurActif].getScore(),joueurs[joueurActif].getFlechette());
00382         }
00383     }
00384 }
```

7.2.3.26 gererMancheTournois()

```
void Darts::gererMancheTournois ( )  [private]
```

Permets de gérer le changement de manche en fonction des fléchettes de chaque joueur.

Définition à la ligne 687 du fichier darts.cpp.

Références calculerMoyenneVoleesTournois(), changementJoueurActifTournoi(), dernierJoueur, gererVoleeMax(), getManche(), jouerSon(), joueurActif, joueurs, NB_FLECHETTE, nouvelleManche(), pointVoleeEnCours, premierJoueur, setManche(), solution, et Solution : :trouverSolution().

Référencé par testerImpactTournois().

```
00688 {
00689     if(joueurs[joueurActif].getFlechette() == 0) //fin de la volées du joueur
00690     {
00691         joueurs[joueurActif].setNbFlechette(NB_FLECHETTE);
00692
00693         pointVoleeEnCours = 0;
00694
00695         gererVoleeMax();
00696         //testerSiJoueurEliminer(); /** @todo changer pour faire gagner l'autre joueur*/
00697
00698         joueurs[joueurActif].addHistoriqueVolees((joueurs[
      joueurActif].getScoreManchePrecedente() - joueurs[joueurActif].getScore()));
00699
00700         if((joueurs[joueurActif].getScoreManchePrecedente() -
      joueurs[joueurActif].getScore()) == 180)
00701             emit jouerSon("180.wav");
00702
00703         joueurs[joueurActif].setScoreManchePrecedente(joueurs[
      joueurActif].getScore());
00704
00705         if(joueurActif == dernierJoueur)  //equivalent a la fin de manche
00706         {
00707             joueurActif = premierJoueur;
00708
00709             setManche(getManche() + 1);
00710             emit changementJoueurActifTournoi();
00711             calculerMoyenneVoleesTournois();
00712             emit nouvelleManche();
00713             solution->trouverSolution(joueurs[
      joueurActif].getScore(),joueurs[joueurActif].getFlechette());
00714         }
00715         else
00716         {
00717             joueurActif++;
00718
00719             emit changementJoueurActifTournoi();
00720             solution->trouverSolution(joueurs[
      joueurActif].getScore(),joueurs[joueurActif].getFlechette());
00721         }
00722     }
00723 }
```

7.2.3.27 gererVoleeMax()

```
void Darts::gererVoleeMax ( )  [private]
```

Teste la volée pour savoir si elle est supérieure à la volée Max.

Définition à la ligne 449 du fichier darts.cpp.

Références getVoleeMax(), joueurActif, joueurs, et setVoleeMax().

Référencé par enleverPointImpact(), gererManche(), gererMancheTournois(), testerImpact(), et testerImpactTournois().

```
00450 {
00451     if((joueurs[joueurActif].getScoreManchePrecedente() -
      joueurs[joueurActif].getScore()) > getVoleeMax())
00452     {
00453         setVoleeMax(joueurs[joueurActif].getScoreManchePrecedente() -
      joueurs[joueurActif].getScore());
00454     }
00455 }
```

7.2.3.28 getDarts()

```
Darts* Darts::getDarts ( ) const
```

7.2.3.29 getDernierJoueur()

```
int Darts::getDernierJoueur ( ) const
```

Retourne le dernier joueur a jouer.

Renvoie

int le dernier joueur a jouer

Définition à la ligne 121 du fichier darts.cpp.

Références dernierJoueur.

Référencé par Ihm : :finirPartie(), Ihm : :initialiserAffichageTournois(), Ihm : :mettreAJourJoueurTournoi(), Ihm : :mettreAJour↩MoyenneVoleeTournois(), et Ihm : :mettreAJourScoreTournois().

```
00122 {
00123     return dernierJoueur;
00124 }
```

7.2.3.30 getJoueurActif()

```
int Darts::getJoueurActif ( ) const
```

Retourne le numéro du joueur actif.

Renvoie

int le numéro du joueur actif

Définition à la ligne 88 du fichier darts.cpp.

Références joueurActif.

Référencé par Ihm : :mettreAJourJoueur(), Ihm : :mettreAJourJoueurTournoi(), et Ihm : :mettreAJourScore().

```
00089 {
00090     return joueurActif;
00091 }
```

7.2.3.31 getListJoueur()

```
QList< Joueur > Darts::getListJoueur ( ) const
```

Retourne une liste des joueurs.

Renvoie

QList<Joueur> la liste des joueurs

Définition à la ligne 66 du fichier darts.cpp.

Références joueurs.

Référencé par Ihm : :finirPartie(), Ihm : :initialiserAffichageTournois(), Ihm : :mettreAJourJoueur(), Ihm : :mettreAJourJoueurTournoi(), Ihm : :mettreAJourMoyenneVolee(), Ihm : :mettreAJourMoyenneVoleeTournois(), Ihm : :mettreAJourScore(), et Ihm : :mettreAJour↩ ScoreTournois().

```
00067 {
00068     return joueurs;
00069 }
```

7.2.3.32 getManche()

```
int Darts::getManche ( ) const
```

Retourne la manche.

Renvoie

int le numéro de manche

Définition à la ligne 44 du fichier darts.cpp.

Références manche.

Référencé par controlerJoueurEliminer(), gererManche(), gererMancheTournois(), Ihm : :initialiserAffichageTournois(), et Ihm↩ : :mettreAJourManche().

```
00045 {
00046     return manche;
00047 }
```

7.2.3.33 getModeDeJeu()

```
QString Darts::getModeDeJeu ( ) const
```

Retourne le mode de jeu.

Renvoie

QString le mode de jeu

Définition à la ligne 132 du fichier darts.cpp.

Références ModeDeJeu.

Référencé par Ihm : :afficherPartie(), et testerModeDeJeu().

```
00133 {
00134     return ModeDeJeu;
00135 }
```

7.2.3.34 getNbVolees()

```
int Darts::getNbVolees ( ) const
```

Retourne le nombre de volées de la partie.

Renvoie

int le nombre de volées de la partie

Définition à la ligne 99 du fichier darts.cpp.

Références nbVolees.

Référencé par Ihm : :finirPartie().

```
00100 {
00101     return nbVolees;
00102 }
```

7.2.3.35 getPointVolees()

```
int Darts::getPointVolees ( ) const
```

Retourne la manche.

Renvoie

int le nombre de point de la manche en cours

Définition à la ligne 55 du fichier darts.cpp.

Références pointVoleeEnCours.

Référencé par Ihm : :mettreAJourMessageStatut().

```
00056 {
00057     return pointVoleeEnCours;
00058 }
```

7.2.3.36 getPremierJoueur()

```
int Darts::getPremierJoueur ( ) const
```

Retourne le premier Joueur a jouer.

Renvoie

int le premier Joueur a jouer

Définition à la ligne 110 du fichier darts.cpp.

Références premierJoueur.

Référencé par Ihm : :finirPartie(), Ihm : :initialiserAffichageTournois(), Ihm : :mettreAJourJoueurTournoi(), Ihm : :mettreAJour↩MoyenneVoleeTournois(), et Ihm : :mettreAJourScoreTournois().

```
00111 {
00112     return premierJoueur;
00113 }
```

7.2.3.37 getSolution()

```
Solution * Darts::getSolution ( ) const
```

retourne l'objet solution

Renvoie

Solution∗ l'objet solution

Définition à la ligne 143 du fichier darts.cpp.

Références solution.

Référencé par Ihm : :initialiserEvenements().

```
00144 {
00145     return solution;
00146 }
```

7.2.3.38 getVoleeMax()

```
int Darts::getVoleeMax ( ) const
```

Retourne la volée max.

Renvoie

int la volée max

Définition à la ligne 77 du fichier darts.cpp.

Références voleeMax.

Référencé par arreterPartie(), enleverPointImpact(), gererVoleeMax(), testerImpact(), et testerImpactTournois().

```
00078 {
00079     return voleeMax;
00080 }
```

7.2.3.39 initialiserFinTournois()

```
void Darts::initialiserFinTournois ( )  [private]
```

methode qui gere la fin du tournois

Définition à la ligne 839 du fichier darts.cpp.

Références finTournois(), joueurActif, joueurs, joueursTournoisEliminer, et NomTournois.

Référencé par gererFinPartieTournois().

```
00840 {
00841     QList<Joueur> joueurTournois;
00842
00843     for(int i = 0 ; i < joueurs.size(); i++)
00844     {
00845         joueurTournois.push_back(joueurs[i]);
00846     }
00847
00848     for(int i = 0 ; i < joueursTournoisEliminer.size(); i++)
00849     {
00850         joueurTournois.push_back(joueursTournoisEliminer[i]);
00851     }
00852
00853     emit finTournois(joueurs[joueurActif].getNom(),
      NomTournois,joueurTournois);
00854 }
```

7.2.3.40 initialiserPartie()

```
void Darts::initialiserPartie (
             QStringList joueurList,
             QString modeJeu )
```

Initialise la partie.

Paramètres

joueurList	QStringList
modeJeu	QString

Définition à la ligne 177 du fichier darts.cpp.

Références afficherNouvellePartie(), changerEtatPartie(), joueurActif, joueurs, ModeDeJeu, NB_FLECHETTE, nbJoueur, reinitialiserPartie(), solution, et Solution : :trouverSolution().

Référencé par Communication : :extraireParametresTrameStart().

```
00178 {
00179     nbJoueur = joueurList.size() - 1;
00180     ModeDeJeu = modeJeu;
00181     qDebug() << Q_FUNC_INFO << "nbJoueur " << nbJoueur << " | Mode De Jeu " <<
      ModeDeJeu;
00182
00183     if(ModeDeJeu.toInt() >= 101 && ModeDeJeu.toInt() <= 1001)
00184     {
00185         for(int i = 1; i < joueurList.size() ; i++)
00186         {
00187             Joueur player(joueurList.at(i), ModeDeJeu.toInt(),
      NB_FLECHETTE);
00188             joueurs.push_back(player);
00189         }
```

```
00190         emit afficherNouvellePartie();
00191         emit changerEtatPartie();
00192     }
00193     else if(ModeDeJeu.contains("_DOUBLE_OUT") && (ModeDeJeu.section("_",0,0).toInt() >= 101 && ModeDeJeu.
      section("_",0,0).toInt() <= 1001))
00194     {
00195         for(int i = 1; i < joueurList.size() ; i++)
00196         {
00197             Joueur player(joueurList.at(i), ModeDeJeu.section("_",0,0).toInt(),
      NB_FLECHETTE);
00198             joueurs.push_back(player);
00199         }
00200         emit afficherNouvellePartie();
00201         emit changerEtatPartie();
00202     }
00203     else
00204     {
00205         qDebug() << Q_FUNC_INFO << "Erreur Mode De Jeu : " << ModeDeJeu;
00206         reinitialiserPartie();
00207         return;
00208     }
00209     solution->trouverSolution(joueurs[joueurActif].getScore(),
      joueurs[joueurActif].getFlechette());
00210 }
```

7.2.3.41 jouerSon

```
void Darts::jouerSon (
            QString son )  [signal]
```

signal émis pour Lancer un son

Référencé par enleverPointImpact(), gererManche(), gererMancheTournois(), et testerPoint().

7.2.3.42 miseAJourMoyenneVolee

```
void Darts::miseAJourMoyenneVolee ( )  [signal]
```

signal émis pour mettre à jour la moyenne des volées

Référencé par calculerMoyenneVolees().

7.2.3.43 miseAJourMoyenneVoleeTournois

```
void Darts::miseAJourMoyenneVoleeTournois ( )  [signal]
```

signal émis pour mettre a jour la moyenne des Volées en mode tournois

Référencé par calculerMoyenneVoleesTournois().

7.2.3.44 miseAJourPoint

```
void Darts::miseAJourPoint ( )  [signal]
```

signal émis pour mettre à jour les points des joueurs

Référencé par receptionnerImpact().

7.2.3.45 miseAJourPointTournois

```
void Darts::miseAJourPointTournois ( )  [signal]
```

signal émis pour mettre a jour les points du tournois

Référencé par receptionnerImpactTournois().

7.2.3.46 nouvelImpact

```
void Darts::nouvelImpact (
            int ,
            int ,
            int  )  [signal]
```

signal émis quand il y a un nouvel Impact

Référencé par receptionnerImpact(), et receptionnerImpactTournois().

7.2.3.47 nouvelleManche

```
void Darts::nouvelleManche ( )  [signal]
```

signal émis quand on change de manche

Référencé par controlerJoueurEliminer(), gererManche(), et gererMancheTournois().

7.2.3.48 receptionnerImpact()

```
void Darts::receptionnerImpact (
            int typePoint,
            int point )
```

Permets de traiter la réception d'impact.

Paramètres

typePoint	
point	

Définition à la ligne 261 du fichier darts.cpp.

Références actualiserCible(), calculerPoints(), joueurActif, joueurs, miseAJourPoint(), NB_FLECHETTE, nouvelImpact(), pointLancer, pointVoleeEnCours, testerImpact(), et testerPoint().

Référencé par Communication : :decomposerTrame().

```
00262 {
00263     calculerPoints(point, typePoint);
```

```
00264
00265     testerPoint(typePoint, point);
00266
00267     pointVoleeEnCours += pointLancer;
00268
00269     if(joueurs[joueurActif].getFlechette() == NB_FLECHETTE)
00270         emit actualiserCible();
00271
00272     qDebug() << Q_FUNC_INFO << joueurs[joueurActif].getNom() << " SCORE : " <<
      joueurs[joueurActif].getScore() - pointLancer << endl;
00273
00274     emit nouvelImpact(typePoint, point, pointLancer);
00275     joueurs[joueurActif].setScore(joueurs[joueurActif].getScore() -
     pointLancer);
00276     testerImpact(typePoint);
00277     emit miseAJourPoint();
00278 }
```

7.2.3.49 receptionnerImpactTournois()

```
void Darts::receptionnerImpactTournois (
            int typePoint,
            int point )
```

methode qui gere le deroulement de la partie tournois

Paramètres

typePoint	
point	

Définition à la ligne 629 du fichier darts.cpp.

Références actualiserCible(), calculerPoints(), joueurActif, joueurs, miseAJourPointTournois(), NB_FLECHETTE, nouvelImpact(), pointLancer, pointVoleeEnCours, testerImpactTournois(), et testerPoint().

Référencé par Communication : :decomposerTrame().

```
00630 {
00631     calculerPoints(point, typePoint);
00632
00633     testerPoint(typePoint, point);
00634
00635     pointVoleeEnCours += pointLancer;
00636
00637     if(joueurs[joueurActif].getFlechette() == NB_FLECHETTE)
00638         emit actualiserCible();
00639
00640     emit nouvelImpact(typePoint, point, pointLancer);
00641
00642     joueurs[joueurActif].setScore(joueurs[joueurActif].getScore() -
     pointLancer);
00643
00644     testerImpactTournois(typePoint);
00645     emit miseAJourPointTournois();
00646 }
```

7.2.3.50 reinitialiserPartie()

```
void Darts::reinitialiserPartie ( )
```

Méthode qui permet de remettre a zero les differents attribut et conteneur pour une nouvelle partie.

Définition à la ligne 217 du fichier darts.cpp.

Références dernierJoueur, joueur, joueurActif, joueurs, joueursTournoisEliminer, manche, ModeDeJeu, nbJoueur, nbVolees, pointLancer, pointVoleeEnCours, premierJoueur, et voleeMax.

Référencé par configurationTournois(), Communication : :decomposerTrame(), Communication : :decomposerTrameTournois(), initialiserPartie(), et Communication : :reamorcerPartie().

```
00218 {
00219     joueurs.clear();
00220     joueur.clear();
00221     joueursTournoisEliminer.clear();
00222     premierJoueur = 0;
00223     dernierJoueur = 1;
00224
00225     ModeDeJeu = "";
00226     nbJoueur = 0;
00227     joueurActif = 0;
00228     manche = 1;
00229     pointLancer = 0;
00230     nbVolees = 0;
00231     voleeMax = 0;
00232     pointVoleeEnCours = 0;
00233 }
```

7.2.3.51 setManche()

```
void Darts::setManche (
            int manche )
```

Permets de mettre à jour le numéro de manche.

Paramètres

manche	le numéro de manche (int)

Définition à la ligne 165 du fichier darts.cpp.

Références manche.

Référencé par controlerJoueurEliminer(), gererFinPartieTournois(), gererManche(), et gererMancheTournois().

```
00166 {
00167     this->manche = manche;
00168 }
```

7.2.3.52 setVoleeMax()

```
void Darts::setVoleeMax (
            int voleeMax )
```

Permets de mettre à jour le volé max.

Paramètres

voleeMax	la volée max(int)

Définition à la ligne 154 du fichier darts.cpp.

Références voleeMax.

Référencé par gererVoleeMax().

```
00155 {
00156     this->voleeMax = voleeMax;
00157 }
```

7.2.3.53 testerImpact()

```
void Darts::testerImpact (
            int typePoint )  [private]
```

Teste si le joueur a gagné

Paramètres

typePoint	

Définition à la ligne 286 du fichier darts.cpp.

Références DOUBLE_POINT, enleverPointImpact(), etatPartieFini(), finPartie(), gererManche(), gererVoleeMax(), getVoleeMax(), joueurActif, joueurs, ModeDeJeu, et nbVolees.

Référencé par receptionnerImpact().

```
00287 {
00288     if(joueurs[joueurActif].getScore()  == 0 && (typePoint ==
      DOUBLE_POINT) && (ModeDeJeu.contains("_DOUBLE_OUT"))) //fin avec double
00289     {
00290         gererVoleeMax();
00291         nbVolees++;
00292         emit finPartie("  Winner " + joueurs[joueurActif].getNom() + "  ",
      getVoleeMax(), false);
00293         emit etatPartieFini();
00294     }
00295     else if(joueurs[joueurActif].getScore()  == 0 && !
      ModeDeJeu.contains("_DOUBLE_OUT"))    //fin sans double
00296     {
00297         gererVoleeMax();
00298         nbVolees++;
00299         emit finPartie("  Winner " + joueurs[joueurActif].getNom() + "  " ,
      getVoleeMax(), false);
00300         emit etatPartieFini();
00301     }
00302     else
00303     {
00304         enleverPointImpact();
00305         gererManche();
00306     }
00307 }
```

7.2.3.54 testerImpactTournois()

```
void Darts::testerImpactTournois (
            int typePoint )  [private]
```

teste l'impact pour le mode tournois

Paramètres

typePoint	

Définition à la ligne 654 du fichier darts.cpp.

Références afficherInfoTournois(), DOUBLE_POINT, enleverPointImpact(), etatPartieAttenteTournois(), finPartie(), gererFinPartieTournois(), gererMancheTournois(), gererVoleeMax(), getVoleeMax(), joueurActif, joueurs, ModeDeJeu, et nbVolees.

Référencé par receptionnerImpactTournois().

```
00655 {
00656     if(joueurs[joueurActif].getScore()  == 0 && (typePoint ==
      DOUBLE_POINT) && (ModeDeJeu.contains("_DOUBLE_OUT"))) //fin avec double
00657     {
00658         gererVoleeMax();
00659         nbVolees++;
00660         emit afficherInfoTournois();
00661         emit finPartie("  " + joueurs[joueurActif].getNom() +" Winner de la
       manche" +"  " , getVoleeMax(), true);
00662         gererFinPartieTournois();
00663         emit etatPartieAttenteTournois();
00664     }
00665     else if(joueurs[joueurActif].getScore()  == 0 && !
      ModeDeJeu.contains("_DOUBLE_OUT"))    //fin sans double
00666     {
00667         gererVoleeMax();
00668         nbVolees++;
00669         emit afficherInfoTournois();
00670         emit finPartie("  " + joueurs[joueurActif].getNom() +" Winner de la
       manche" +"  " , getVoleeMax(), true);
00671         gererFinPartieTournois();
00672         emit etatPartieAttenteTournois();
00673     }
00674     else
00675     {
00676         enleverPointImpact();
00677         gererMancheTournois();
00678     }
00679 }
```

7.2.3.55 testerModeDeJeu()

```
QString Darts::testerModeDeJeu ( )
```

Méthode qui teste le mode de jeu.

Renvoie

QString

Définition à la ligne 547 du fichier darts.cpp.

Références getModeDeJeu().

Référencé par Communication : :extraireParametresTrameRegle(), et Communication : :extraireParametresTrameStart().

```
00548 {
00549     QString regle ="";
00550
00551     if(getModeDeJeu().contains("DOUBLE_OUT"))
00552     {
00553         regle = "DOUBLE_OUT";
00554     }
00555     else
00556     {
00557         regle = "SANS_DOUBLE_OUT";
00558     }
00559     return regle;
00560 }
```

7.2.3.56 testerNombreJoueurRestand()

```
void Darts::testerNombreJoueurRestand ( )  [private]
```

Teste s'il reste qu'un joueur n'était pas éliminé

Définition à la ligne 494 du fichier darts.cpp.

Références arreterPartie(), et joueurs.

Référencé par gererManche().

```
00495 {
00496     int eliminer = 0;
00497
00498     for(int i = 0; i <= joueurs.size() - 1; i++)
00499     {
00500         if(joueurs[i].getEliminer())
00501         {
00502             eliminer++;
00503         }
00504     }
00505
00506     if(eliminer == joueurs.size() - 1 && joueurs.size() != 1)
00507     {
00508         arreterPartie();
00509     }
00510 }
```

7.2.3.57 testerPoint()

```
void Darts::testerPoint (
            int typePoint,
            int point )  [private]
```

Méthode qui teste les Impact pour savoir quel son jouer.

Paramètres

typePoint	
point	

Définition à la ligne 242 du fichier darts.cpp.

Références BULL, DOUBLE_POINT, jouerSon(), et ZERO_POINT.

Référencé par receptionnerImpact(), et receptionnerImpactTournois().

```
00243 {
00244     if(typePoint == DOUBLE_POINT && point == BULL)
00245     {
00246         jouerSon("D25.wav");
00247     }
00248     else if(typePoint == ZERO_POINT)
00249     {
00250         jouerSon("out.wav");
00251     }
00252 }
```

7.2.3.58 testerSiJoueurEliminer()

```
void Darts::testerSiJoueurEliminer ( )  [private]
```

Teste si le joueur est à 1 point à la fin de la manche.

Définition à la ligne 414 du fichier darts.cpp.

Références joueurActif, et joueurs.

Référencé par gererManche().

```
00415 {
00416     if(joueurs[joueurActif].getScore() == 1)    //joueur eliminé si tomber à 1 point
00417     {
00418         joueurs[joueurActif].setEliminer(true);
00419     }
00420 }
```

7.2.3.59 voleeAnnulee

```
void Darts::voleeAnnulee ( )  [signal]
```

signal émis quand la volées est annulé

Référencé par enleverPointImpact().

7.2.4 Documentation des données membres

7.2.4.1 dernierJoueur

```
int Darts::dernierJoueur  [private]
```

contient le dernier joueur du tournois a jouer

Définition à la ligne 102 du fichier darts.h.

Référencé par calculerMoyenneVoleesTournois(), configurationTournois(), gererFinPartieTournois(), gererMancheTournois(), getDernierJoueur(), et reinitialiserPartie().

7.2.4.2 joueur

```
QStringList Darts::joueur  [private]
```

contient les noms des differents joueur

Définition à la ligne 91 du fichier darts.h.

Référencé par reinitialiserPartie().

7.2.4.3 joueurActif

```
int Darts::joueurActif  [private]
```

contient le numero du joueur en train de jouer

Définition à la ligne 93 du fichier darts.h.

Référencé par configurationTournois(), controlerJoueurEliminer(), enleverPointImpact(), gererFinPartieTournois(), gererManche(), gererMancheTournois(), gererVoleeMax(), getJoueurActif(), initialiserFinTournois(), initialiserPartie(), receptionnerImpact(), receptionnerImpactTournois(), reinitialiserPartie(), testerImpact(), testerImpactTournois(), et testerSiJoueurEliminer().

7.2.4.4 joueurs

```
QList<Joueur> Darts::joueurs  [private]
```

contient des objets joueurs

Définition à la ligne 89 du fichier darts.h.

Référencé par calculerGagnant(), calculerMoyenneVolees(), calculerMoyenneVoleesTournois(), configurationTournois(), controler↩JoueurEliminer(), enleverPointImpact(), estDernier(), gererFinPartieTournois(), gererManche(), gererMancheTournois(), gererVolee↩Max(), getListJoueur(), initialiserFinTournois(), initialiserPartie(), receptionnerImpact(), receptionnerImpactTournois(), reinitialiser↩Partie(), testerImpact(), testerImpactTournois(), testerNombreJoueurRestand(), et testerSiJoueurEliminer().

7.2.4.5 joueursTournoisEliminer

```
QList<Joueur> Darts::joueursTournoisEliminer  [private]
```

contient des objets joueurs eliminer pendant le tournois

Définition à la ligne 90 du fichier darts.h.

Référencé par gererFinPartieTournois(), initialiserFinTournois(), et reinitialiserPartie().

7.2.4.6 manche

```
int Darts::manche  [private]
```

contient le numero de la manche actuel

Définition à la ligne 94 du fichier darts.h.

Référencé par getManche(), reinitialiserPartie(), et setManche().

7.2.4.7 ModeDeJeu

```
QString Darts::ModeDeJeu  [private]
```

contient le mode de jeu en cours

Définition à la ligne 98 du fichier darts.h.

Référencé par configurationTournois(), gererFinPartieTournois(), getModeDeJeu(), initialiserPartie(), reinitialiserPartie(), testerImpact(), et testerImpactTournois().

7.2.4.8 nbJoueur

```
int Darts::nbJoueur  [private]
```

contient le nombre de joueur

Définition à la ligne 92 du fichier darts.h.

Référencé par configurationTournois(), controlerJoueurEliminer(), gererManche(), initialiserPartie(), et reinitialiserPartie().

7.2.4.9 nbVolees

```
int Darts::nbVolees  [private]
```

contient le nombre de Volées de la partie en cours

Définition à la ligne 97 du fichier darts.h.

Référencé par enleverPointImpact(), gererFinPartieTournois(), getNbVolees(), reinitialiserPartie(), testerImpact(), et testerImpactTournois().

7.2.4.10 NomTournois

```
QString Darts::NomTournois  [private]
```

contient le nom du tournois

Définition à la ligne 99 du fichier darts.h.

Référencé par configurationTournois(), gererFinPartieTournois(), et initialiserFinTournois().

7.2.4.11 pointLancer

```
int Darts::pointLancer  [private]
```

contient les point associer l'impact de la fleche

Définition à la ligne 95 du fichier darts.h.

Référencé par calculerPoints(), receptionnerImpact(), receptionnerImpactTournois(), et reinitialiserPartie().

7.2.4.12 pointVoleeEnCours

```
int Darts::pointVoleeEnCours  [private]
```

contient le score de la Volées en cours

Définition à la ligne 100 du fichier darts.h.

Référencé par gererManche(), gererMancheTournois(), getPointVolees(), receptionnerImpact(), receptionnerImpactTournois(), et reinitialiserPartie().

7.2.4.13 premierJoueur

```
int Darts::premierJoueur  [private]
```

contient le premier joueur du tournois a jouer

Définition à la ligne 101 du fichier darts.h.

Référencé par calculerMoyenneVoleesTournois(), configurationTournois(), gererFinPartieTournois(), gererMancheTournois(), getPremierJoueur(), et reinitialiserPartie().

7.2.4.14 solution

```
Solution* Darts::solution  [private]
```

Association vers l'objet solution.

Définition à la ligne 88 du fichier darts.h.

Référencé par configurationTournois(), Darts(), enleverPointImpact(), gererManche(), gererMancheTournois(), getSolution(), et initialiserPartie().

7.2.4.15 voleeMax

```
int Darts::voleeMax  [private]
```

contient la volées Max

Définition à la ligne 96 du fichier darts.h.

Référencé par getVoleeMax(), reinitialiserPartie(), et setVoleeMax().

La documentation de cette classe a été générée à partir des fichiers suivants :

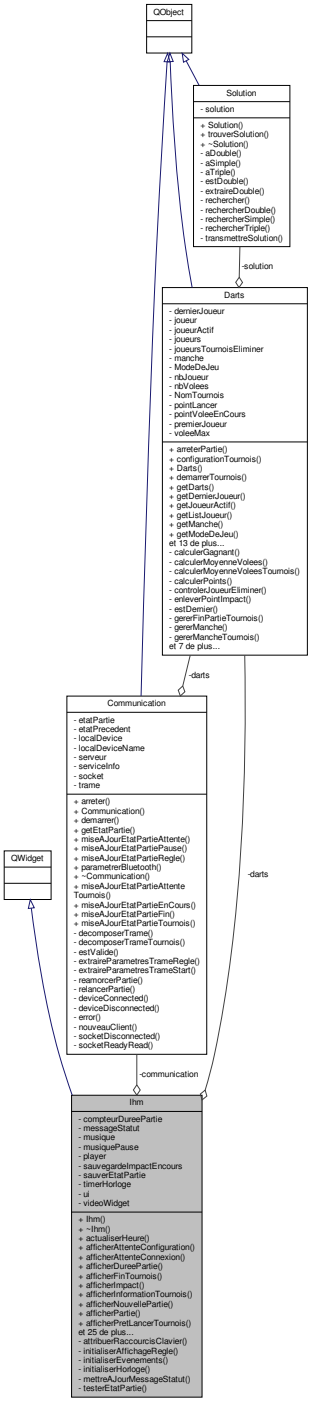
- darts.h
- darts.cpp

7.3 Référence de la classe Ihm

Déclaration de la classe Ihm (Module Ecran-DARTS)

`#include <ihm.h>`

Graphe de collaboration de Ihm :



Connecteurs publics

— void actualiserHeure ()

Méthode qui met a jour l'heure sur l'application.

— void afficherAttenteConfiguration ()
Méthode qui permet de mettre a jour le message de status "nouvelle appareil connecté".

— void afficherAttenteConnexion ()
Méthode qui permet de mettre a jour le message de status "appareil deconnecté".

— void afficherDureePartie ()
Affiche la durée d'une Seance(slot)

— void afficherFinTournois (QString nomGagnant, QString nomTournois, QList< Joueur > joueurs)
Méthode qui gère l'affichage quand le tournoi est terminé

— void afficherImpact (int typePoint, int point)
Méthode qui affiche la cible correspondante à l'impact (si fichier Image Impact et disponible) et les points de cette Impact.

— void afficherInformationTournois ()
Méthode qui gère l'affichage des informations du tournoi.

— void afficherNouvellePartie ()
Méthode qui met à jour l'affichage pour lancer une nouvelle partie.

— void afficherPartie ()
Méthode qui met a jour le mode de jeu et la page actif.

— void afficherPretLancerTournois ()

— void afficherVoleeAnnulee ()
Méthode qui affiche le message de statut "volée annulée".

— void allerPage (Ihm : :Page page)
Méthode qui permet de changer de QStackedWidget avec la Précédente.

— void allerPagePrecedente ()
Méthode qui permet de changer de QStackedWidget avec la Précédente.

— void allerPageSuivante ()
Méthode qui permet de changer de QStackedWidget avec la suivante.

— void error (QMediaPlayer : :Error error)
Méthode appelée quand il y a une erreur de vidéo.

— void fermerApplication ()
Méthode qui permet de quitter l'application.

— void finirPartie (QString gagnant, int voleeMaxJoueur, bool tournois)
Méthode qui met à jour l'affichage quand la partie est fini.

— void initialiserAffichageTournois (QString modeJeu, QString nomTournois)
Méthode qui initialise l'affichage du tournois.

— void jouerSon (QString son)
Méthode qui permet de jouer un son.

— void lancerRegle (QString regle)
Méthode qui lance la vidéo explicative des regles suivant le type de jeu.

— void lancerTournois ()
methode qui affiche le l'ecran de tournois

— void mettreAJourCible ()
Méthode qui reinitialise l'affichage de la cible.

— void mettreAJourJoueur ()
Méthode qui initialise l'affichage du mode et des joueurs de la partie.

— void mettreAJourJoueurTournoi ()
Méthode qui initialise l'affichage du mode et des joueurs de la partie.

— void mettreAJourManche ()
Méthode qui met à jour le numero de la manche.

— void mettreAJourMessageStatut (QString)
Méthode qui met à jour le message de statut.

— void mettreAJourMoyenneVolee ()
Méthode qui initialise l'affichage du mode et des joueurs de la partie.

— void mettreAJourMoyenneVoleeTournois ()
Méthode qui met a jour la moyenne des Volee du joueur.

— void mettreAJourScore ()
Méthode qui met à jour le score dans L'ihm.

— void mettreAJourScoreTournois ()
methode qui met a jour le score des joueurs des tournois

— void mettreAJoursolution (QString solution)
Affiche les solutions possibles pour finir la parties.

— void mettrePausePartie ()
Mets en pause le chronométrage de la partiee.

— void relancerpartie ()
relancer le chronométrage de la partie

— void stateChanged (QMediaPlayer : :State state)
Méthode appelée quand l'état de la vidéo change.

— void StopperLectureRegle ()
methode qui stop la lecture de la musique

Fonctions membres publiques

— Ihm (QWidget *parent=nullptr)

Constructeur de la classe Ihm.

— ~Ihm ()
Destructeur de la classe Ihm.

Types privés

— enum Page {
PageAttente = 0, PageRegle, PageJeu, PageStatistique,
PageTournois, PageFinTournois, NbPages }
Définit les différentes pages de l'IHM.

Fonctions membres privées

— void attribuerRaccourcisClavier ()
Méthode qui initialise les raccourcis clavier.
— void initialiserAffichageRegle ()
Méthode qui initialise l'affichage vidéo des règles.
— void initialiserEvenements ()
Méthode qui initialise les connexion signals/slots de Qt.
— void initialiserHorloge ()
initialise l'horloge pour un affichage périodique
— void mettreAJourMessageStatut (int typePoint, int point)
Méthode qui met à jour le message de statut de la volée en cours.
— void testerEtatPartie ()
Méthode appelée pour remettre l'état dans lequel était la partie avant l'affichage des règles.

Attributs privés

— Communication * communication
objet communication
— int compteurDureePartie
compteur de secondes pour la durée d'une séance
— Darts * darts
objet darts
— QString messageStatut
contient le message de statut qui est affiché
— QSound musique
objet musique
— QSound musiquePause
objet musiquePause
— QMediaPlayer * player
objet player
— QPixmap sauvegardeImpactEncours
sauvegarde le QPixmap de l'état de la cible
— int sauverEtatPartie
Contient l'état de la partie avant l'affichage des règles.
— QTimer * timerHorloge
objet timerHorloge
— Ui : :Ihm * ui
object de notre Ihm
— QVideoWidget * videoWidget
objet videoWidget

7.3.1 Description détaillée

Déclaration de la classe Ihm (Module Ecran-DARTS)

Cette classe s'occupe de l'affichage sur l'écran

Définition à la ligne 43 du fichier ihm.h.

7.3.2 Documentation des énumérations membres

7.3.2.1 Page

```
enum Ihm::Page  [private]
```

Définit les différentes pages de l'IHM.

Valeurs énumérées

PageAttente	
PageRegle	
PageJeu	
PageStatistique	
PageTournois	
PageFinTournois	
NbPages	

Définition à la ligne 72 du fichier ihm.h.

```
00073     {
00074         PageAttente = 0,
00075         PageRegle,
00076         PageJeu,
00077         PageStatistique,
00078         PageTournois,
00079         PageFinTournois,
00080         NbPages
00081     };
```

7.3.3 Documentation des constructeurs et destructeur

7.3.3.1 Ihm()

```
Ihm::Ihm (
            QWidget * parent = nullptr )  [explicit]
```

Constructeur de la classe Ihm.

Paramètres

parent	

Définition à la ligne 26 du fichier ihm.cpp.

Références afficherNouvellePartie(), attribuerRaccourcisClavier(), communication, darts, Communication : :demarrer(), initialiserAffichageRegle(), initialiserEvenements(), initialiserHorloge(), et ui.

```
00026                       :
00027     QWidget(parent),
```

```
00028     ui(new Ui::Ihm),
00029     musique(qApp->applicationDirPath() + CHEMIN_FICHIER_MUSIQUE + "music.wav",
      this),
00030     musiquePause(qApp->applicationDirPath() + CHEMIN_FICHIER_MUSIQUE + "
      pause.wav",this),
00031     compteurDureePartie(0),
00032     messageStatut("Volée ")
00033 {
00034     ui->setupUi(this);
00035     qDebug() << Q_FUNC_INFO;
00036
00037     darts = new Darts(this);
00038     communication = new Communication(darts, this);
00039
00040     // Initialiser l'horloge
00041     initialiserHorloge();
00042
00043     // Raccourcis Quitter/ChangerPage (mode debug)
00044     attribuerRaccourcisClavier();
00045
00046     // Démarrer la communication bluetooth
00047     communication->demarrer();
00048
00049     // Initialiser les connexions signal/slot
00050     initialiserEvenements();
00051
00052     //configuration affichage des regles
00053     initialiserAffichageRegle();
00054
00055     //Afficher la page d'attente
00056     afficherNouvellePartie();
00057 }
```

7.3.3.2 ~Ihm()

`Ihm::~Ihm ( )`

Destructeur de la classe Ihm.

Définition à la ligne 64 du fichier ihm.cpp.

Références ui.

```
00065 {
00066     delete ui;
00067     qDebug() << Q_FUNC_INFO;
00068 }
```

7.3.4 Documentation des fonctions membres

7.3.4.1 actualiserHeure

`void Ihm::actualiserHeure ( )  [slot]`

Méthode qui met a jour l'heure sur l'application.

Définition à la ligne 136 du fichier ihm.cpp.

Références ui.

Référencé par initialiserHorloge().

```
00137 {
00138     QString affichageHeure;
00139     QTime heure = QTime::currentTime();
00140     affichageHeure = "<font color=\"#6D2B6B\">" + heure.toString("hh : mm ") + "</font>";
00141     ui->labelHeureAttente->setText(affichageHeure);
00142     ui->labelHeureStatistique->setText(affichageHeure);
00143     ui->labelHeureTournois->setText(heure.toString("hh : mm "));
00144 }
```

7.3.4.2 afficherAttenteConfiguration

```
void Ihm::afficherAttenteConfiguration ( )  [slot]
```

Méthode qui permet de mettre a jour le message de status "nouvelle appareil connecté".

Définition à la ligne 502 du fichier ihm.cpp.

Références ui.

Référencé par initialiserEvenements().

```
00503 {
00504     ui->labelStatutAttente->setText(" Attente configuration de la partie ");
00505 }
```

7.3.4.3 afficherAttenteConnexion

```
void Ihm::afficherAttenteConnexion ( )  [slot]
```

Méthode qui permet de mettre a jour le message de status "appareil deconnecté".

Définition à la ligne 512 du fichier ihm.cpp.

Références ui.

Référencé par initialiserEvenements().

```
00513 {
00514     ui->labelStatutAttente->setText(" En attente de connexion ");
00515 }
```

7.3.4.4 afficherDureePartie

```
void Ihm::afficherDureePartie ( )  [slot]
```

Affiche la durée d'une Seance(slot)

Définition à la ligne 533 du fichier ihm.cpp.

Références compteurDureePartie, et ui.

Référencé par afficherPartie(), finirPartie(), lancerTournois(), mettrePausePartie(), et relancerpartie().

```
00534 {
00535     QTime duree(0, 0);
00536     compteurDureePartie++;
00537     QTime dureeSeance = duree.addSecs(compteurDureePartie);
00538     if(compteurDureePartie >= 3600)
00539     {
00540         ui->ecranPartie->setStyleSheet("
      QWidget#ecranPartie{background-image:url(:/ressources/backgroundHeure.jpg);}");
00541         ui->labelTempsPartie->setText(dureeSeance.toString("hh : mm : ss"));
00542         ui->tempsPartie->setText(dureeSeance.toString("hh : mm : ss"));
00543         ui->tournoisChrono->setText(dureeSeance.toString("hh : mm : ss"));
00544     }
00545     else
00546     {
00547         ui->labelTempsPartie->setText(dureeSeance.toString("mm : ss"));
00548         ui->tempsPartie->setText(dureeSeance.toString("mm : ss"));
00549         ui->tournoisChrono->setText(dureeSeance.toString("mm : ss"));
00550     }
00551 }
```

7.3.4.5 afficherFinTournois

```
void Ihm::afficherFinTournois (
            QString nomGagnant,
            QString nomTournois,
            QList< Joueur > joueurs )  [slot]
```

Méthode qui gère l'affichage quand le tournoi est terminé

Définition à la ligne 846 du fichier ihm.cpp.

Références allerPage(), PageFinTournois, et ui.

Référencé par initialiserEvenements().

```
00847 {
00848     QString joueurTournois = "";
00849
00850     for(int i = 0 ; i < joueurs.size(); i++)
00851     {
00852
00853         if(i == 0)
00854         {
00855             joueurTournois = "          1er  " + joueurs[i].getNom() + "\n";
00856         }
00857         else
00858         {
00859             joueurTournois = joueurTournois +"          "+ QString::number(i + 1) + "ème  " + joueurs[i].
      getNom() + "\n";
00860         }
00861     }
00862
00863     ui->winnerTournois->setText(" " + nomGagnant + " grand(e) gagnant(e) du tournoi " + nomTournois +"
      ");
00864     ui->recapPlaceTournois->setText(joueurTournois);
00865     allerPage(Ihm::PageFinTournois);
00866 }
```

7.3.4.6 afficherImpact

```
void Ihm::afficherImpact (
            int typePoint,
            int point )  [slot]
```

Méthode qui affiche la cible correspondante à l'impact (si fichier Image Impact et disponible) et les points de cette Impact.

Paramètres

typePoint	
point	

Définition à la ligne 202 du fichier ihm.cpp.

Références mettreAJourMessageStatut(), et ui.

Référencé par initialiserEvenements().

```
00203 {
00204     if(QFileInfo(qApp->applicationDirPath() + "/impact/IMPACT_" + QString::number(typePoint) + "_" +
      QString::number(point) + ".png").exists())       //test si l'image existe
00205     {
00206         QImage impact(qApp->applicationDirPath() + "/impact/IMPACT_" + QString::number(typePoint) + "_" +
```

```
      QString::number(point) + ".png");
00207
00208         QPixmap cibleImpacte = ui->labelVisualisationimpact->pixmap()->copy(); // on récupère l'image
       précédente;
00209
00210         QPainter p(&cibleImpacte);
00211
00212         p.drawImage(QPoint(0, 0), impact);
00213
00214         p.end();
00215
00216         ui->labelVisualisationimpact->setPixmap(cibleImpacte);
00217         ui->ImpactCibleTournois->setPixmap(cibleImpacte);
00218     }
00219
00220     mettreAJourMessageStatut(typePoint, point);
00221 }
```

7.3.4.7 afficherInformationTournois

```
void Ihm::afficherInformationTournois ( )  [slot]
```

Méthode qui gère l'affichage des informations du tournoi.

Définition à la ligne 873 du fichier ihm.cpp.

Références ui.

Référencé par initialiserEvenements().

```
00874 {
00875     ui->labelINfoMatch->setVisible(true);
00876 }
```

7.3.4.8 afficherNouvellePartie

```
void Ihm::afficherNouvellePartie ( )  [slot]
```

Méthode qui met à jour l'affichage pour lancer une nouvelle partie.

Définition à la ligne 406 du fichier ihm.cpp.

Références allerPage(), mettreAJourCible(), musique, musiquePause, PageAttente, player, et ui.

Référencé par Ihm(), et initialiserEvenements().

```
00407 {
00408      player->stop();
00409      allerPage(Ihm::PageAttente);
00410      ui->manche->setText("1");
00411      ui->nomJoueur->setText("");
00412      ui->scoreActuel->setText("");
00413      ui->typeJeu->setText("");
00414      ui->winnerPartie->setText("Winner ....");
00415      ui->labelStatut->setText("");
00416      ui->moyenneVolee->setText("");
00417      ui->moyenneVolee2->setText("");
00418      ui->nbVolees->setText("");
00419      ui->voleeMax->setText("");
00420      ui->moyenneVolees->setText("");
00421      ui->labelMoyenneVolees->setVisible(false);
00422      ui->lineScoreActuel->setVisible(false);
00423      ui->lineSeparateurMoyenne->setVisible(false);
00424      ui->labelMoyenneVoleesStatistique->setVisible(false);
00425      ui->ecranPartie->setStyleSheet("
      QWidget#ecranPartie{background-image:url(:/ressources/background.jpg);}");
```

```
00426       mettreAJourCible();
00427       ui->labelINfoMatch->setVisible(false);
00428
00429       //ecran Tournois
00430       ui->tournoisNom->setText("----------");
00431       ui->modeDeJeuTournois->setText("");
00432       ui->nomJoueurTournois1->setText("");
00433       ui->nomJoueurTournois2->setText("");
00434       ui->scoreJoueurTournois1->setText("\n\n");
00435       ui->scoreJoueurTournois2->setText("\n\n");
00436       ui->moyenneJoueurTournois1->setText("\n\n" );
00437       ui->moyenneJoueurTournois2->setText("\n\n" );
00438       ui->tournoisManche->setText("Manche 1");
00439       ui->statutImpactTournois->setText("");
00440
00441
00442       // configurer la musique
00443       musique.setLoops(QSound::Infinite);
00444       musiquePause.setLoops(QSound::Infinite);
00445       musiquePause.stop();
00446       // jouer la musique
00447       musique.play();
00448 }
```

7.3.4.9 afficherPartie

```
void Ihm::afficherPartie ( )  [slot]
```

Méthode qui met a jour le mode de jeu et la page actif.

Définition à la ligne 348 du fichier ihm.cpp.

Références afficherDureePartie(), allerPage(), compteurDureePartie, darts, Darts : :getModeDeJeu(), mettreAJourJoueur(), mettre↩ AJourScore(), musique, PageJeu, timerHorloge, et ui.

Référencé par initialiserEvenements().

```
00349 {
00350     musique.stop();
00351
00352     ui->typeJeu->setText(darts->getModeDeJeu());
00353
00354     mettreAJourJoueur();
00355     compteurDureePartie = 0;
00356     connect(timerHorloge, SIGNAL(timeout()),this,SLOT(
      afficherDureePartie())); // Pour le comptage et l'affichage de la durée d'une séance
00357
00358     allerPage(Ihm::PageJeu);
00359
00360     mettreAJourScore();
00361 }
```

7.3.4.10 afficherPretLancerTournois

```
void Ihm::afficherPretLancerTournois ( )  [slot]
```

Définition à la ligne 522 du fichier ihm.cpp.

Références ui.

Référencé par initialiserAffichageTournois().

```
00523 {
00524     ui->labelStatutAttente->setText(" Prêt à lancer le tournoi ");
00525 }
```

7.3.4.11 afficherVoleeAnnulee

```
void Ihm::afficherVoleeAnnulee ( )  [slot]
```

Méthode qui affiche le message de statut "volée annulée".

Définition à la ligne 368 du fichier ihm.cpp.

Références ui.

Référencé par initialiserEvenements().

```
00369 {
00370     ui->labelStatut->setText("Volée annulée !");
00371     ui->statutImpactTournois->setText("Volée annulée !");
00372 }
```

7.3.4.12 allerPage

```
void Ihm::allerPage (
            Ihm::Page page )  [slot]
```

Méthode qui permet de changer de QStackedWidget avec la Précédente.

Paramètres

page	la page du QStackedWidget à afficher

Définition à la ligne 468 du fichier ihm.cpp.

Références ui.

Référencé par afficherFinTournois(), afficherNouvellePartie(), afficherPartie(), finirPartie(), lancerRegle(), lancerTournois(), et testerEtatPartie().

```
00469 {
00470     ui->ecranDarts->setCurrentIndex(page);
00471 }
```

7.3.4.13 allerPagePrecedente

```
void Ihm::allerPagePrecedente ( )  [slot]
```

Méthode qui permet de changer de QStackedWidget avec la Précédente.

Définition à la ligne 478 du fichier ihm.cpp.

Références NbPages, et ui.

Référencé par attribuerRaccourcisClavier().

```
00479 {
00480     int ecranCourant = ui->ecranDarts->currentIndex();
00481     int ecranPrecedent = (ecranCourant-1)%int(Ihm::NbPages);
00482     if(ecranPrecedent == -1)
00483         ecranPrecedent = NbPages-1;
00484     ui->ecranDarts->setCurrentIndex(ecranPrecedent);
00485 }
```

7.3.4.14 allerPageSuivante

```
void Ihm::allerPageSuivante ( )  [slot]
```

Méthode qui permet de changer de QStackedWidget avec la suivante.

Définition à la ligne 455 du fichier ihm.cpp.

Références NbPages, et ui.

Référencé par attribuerRaccourcisClavier().

```
00456 {
00457     int ecranCourant = Page(ui->ecranDarts->currentIndex());
00458     int ecranSuivant = (ecranCourant+1)%int(Ihm::NbPages);
00459     ui->ecranDarts->setCurrentIndex(ecranSuivant);
00460 }
```

7.3.4.15 attribuerRaccourcisClavier()

```
void Ihm::attribuerRaccourcisClavier ( )  [private]
```

Méthode qui initialise les raccourcis clavier.

Définition à la ligne 112 du fichier ihm.cpp.

Références allerPagePrecedente(), allerPageSuivante(), et fermerApplication().

Référencé par Ihm().

```
00113 {
00114     QAction *quitter = new QAction(this);
00115     quitter->setShortcut(QKeySequence(QKeySequence(Qt::Key_Up)));       //fleche du haut pour quitter
      l'application
00116     addAction(quitter);
00117     connect(quitter, SIGNAL(triggered()), this, SLOT(fermerApplication()));   // Pour
      fermer l'application
00118
00119 #ifndef QT_NO_DEBUG_OUTPUT
00120     QAction *actionAllerDroite = new QAction(this);
00121     actionAllerDroite->setShortcut(QKeySequence(Qt::Key_Right));
00122     addAction(actionAllerDroite);
00123     connect(actionAllerDroite, SIGNAL(triggered()), this, SLOT(allerPageSuivante()));//
      Pour passer à l'écran suivant
00124     QAction *actionAllerGauche = new QAction(this);
00125     actionAllerGauche->setShortcut(QKeySequence(Qt::Key_Left));
00126     addAction(actionAllerGauche);
00127     connect(actionAllerGauche, SIGNAL(triggered()), this, SLOT(
     allerPagePrecedente()));// Pour revenir à l'écran précédent
00128 #endif
00129 }
```

7.3.4.16 error

```
void Ihm::error (
            QMediaPlayer::Error error )  [slot]
```

Méthode appelée quand il y a une erreur de vidéo.

Paramètres

error	

Définition à la ligne 755 du fichier ihm.cpp.

Références player, et testerEtatPartie().

Référencé par initialiserAffichageRegle().

```
00756 {
00757     qDebug() << Q_FUNC_INFO << player->errorString() << error;
00758     testerEtatPartie();
00759 }
```

7.3.4.17 fermerApplication

```
void Ihm::fermerApplication ( )  [slot]
```

Méthode qui permet de quitter l'application.

Définition à la ligne 492 du fichier ihm.cpp.

Référencé par attribuerRaccourcisClavier().

```
00493 {
00494     this->close();
00495 }
```

7.3.4.18 finirPartie

```
void Ihm::finirPartie (
            QString gagnant,
            int voleeMaxJoueur,
            bool tournois )  [slot]
```

Méthode qui met à jour l'affichage quand la partie est fini.

Paramètres

gagnant	
voleeMaxJoueur	

Définition à la ligne 381 du fichier ihm.cpp.

Références afficherDureePartie(), allerPage(), darts, Darts : :getDernierJoueur(), Darts : :getListJoueur(), Darts : :getNbVolees(), Darts : :getPremierJoueur(), musique, PageStatistique, player, timerHorloge, et ui.

Référencé par initialiserEvenements().

```
00382 {
00383     player->stop();
00384     musique.play();
00385
00386     if(tournois == true)
00387     {
00388         ui->labelMoyenneVoleesStatistique->setVisible(true);
00389         ui->moyenneVolees->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom() + "  " + QString::number(
      darts->getListJoueur()[darts->getPremierJoueur()].getMoyenneVolee())
       + "\n" + darts->getListJoueur()[darts->getDernierJoueur()].getNom()
      + "  " +QString::number(darts->getListJoueur()[darts->
      getDernierJoueur()].getMoyenneVolee()));
00390         ui->moyenneVolees->setVisible(true);
00391     }
00392
00393     disconnect(timerHorloge, SIGNAL(timeout()), this, SLOT(
      afficherDureePartie())); // Pour le comptage et l'affichage de la durée d'une séance
00394     ui->winnerPartie->setText(gagnant);
00395     ui->voleeMax->setText(QString::number(voleeMaxJoueur) + " points");
00396     ui->nbVolees->setText(QString::number(darts->getNbVolees()));
00397     //communication->miseAJourEtatPartieFin();
00398     allerPage(Ihm::PageStatistique);
00399 }
```

7.3.4.19 initialiserAffichageRegle()

```
void Ihm::initialiserAffichageRegle ( )  [private]
```

Méthode qui initialise l'affichage vidéo des règles.

Définition à la ligne 660 du fichier ihm.cpp.

Références error(), player, stateChanged(), ui, et videoWidget.

Référencé par Ihm().

```
00661 {
00662     player = new QMediaPlayer;
00663
00664     videoWidget = new QVideoWidget(this);
00665     ui->verticalLayoutRegle->addWidget(videoWidget);
00666     player->setVideoOutput(videoWidget);
00667
00668     connect(player, SIGNAL(stateChanged(QMediaPlayer::State)), this, SLOT(
      stateChanged(QMediaPlayer::State)));
00669     connect(player, SIGNAL(error(QMediaPlayer::Error)), this, SLOT(
      error(QMediaPlayer::Error)));
00670 }
```

7.3.4.20 initialiserAffichageTournois

```
void Ihm::initialiserAffichageTournois (
            QString modeJeu,
            QString nomTournois )  [slot]
```

Méthode qui initialise l'affichage du tournois.

Définition à la ligne 766 du fichier ihm.cpp.

Références afficherPretLancerTournois(), darts, Darts : :getDernierJoueur(), Darts : :getListJoueur(), Darts : :getManche(), Darts : :getPremierJoueur(), et ui.

Référencé par initialiserEvenements().

```
00767 {
00768     ui->tournoisNom->setText("Tournoi " + nomTournois);
00769     ui->modeDeJeuTournois->setText(modeJeu);
00770
00771     ui->nomJoueurTournois1->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom());
00772     ui->nomJoueurTournois2->setText(darts->getListJoueur()[
      darts->getDernierJoueur()].getNom());
00773
00774     ui->scoreJoueurTournois1->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom() + "\n\n" + QString::number(
      darts->getListJoueur()[darts->getPremierJoueur()].getScore()));
00775     ui->scoreJoueurTournois2->setText(darts->getListJoueur()[
      darts->getDernierJoueur()].getNom() + "\n\n" + QString::number(
      darts->getListJoueur()[darts->getDernierJoueur()].getScore()));
00776
00777     ui->moyenneJoueurTournois1->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom() + "\n\n0" );
00778     ui->moyenneJoueurTournois2->setText(darts->getListJoueur()[
      darts->getDernierJoueur()].getNom() + "\n\n0" );
00779
00780     ui->tournoisManche->setText("Manche " + QString::number(darts->
      getManche()));
00781
00782     afficherPretLancerTournois();
00783 }
```

7.3.4.21 initialiserEvenements()

```
void Ihm::initialiserEvenements ( )  [private]
```

Méthode qui initialise les connexion signals/slots de Qt.

Définition à la ligne 75 du fichier ihm.cpp.

Références afficherAttenteConfiguration(), afficherAttenteConnexion(), afficherFinTournois(), afficherImpact(), afficherInformation↩ Tournois(), afficherNouvellePartie(), afficherPartie(), afficherVoleeAnnulee(), communication, darts, finirPartie(), Darts : :getSolution(), initialiserAffichageTournois(), jouerSon(), lancerRegle(), lancerTournois(), mettreAJourCible(), mettreAJourJoueur(), mettreA↩ JourJoueurTournoi(), mettreAJourManche(), mettreAJourMessageStatut(), mettreAJourMoyenneVolee(), mettreAJourMoyenne↩ VoleeTournois(), mettreAJourScore(), mettreAJourScoreTournois(), mettreAJoursolution(), mettrePausePartie(), relancerpartie(), et StopperLectureRegle().

Référencé par Ihm().

```
00076 {
00077     connect(communication, SIGNAL(appareilConnecter()) , this, SLOT(
      afficherAttenteConfiguration()));
00078     connect(communication, SIGNAL(afficherAttenteConnexion()), this,
      SLOT(afficherAttenteConnexion()));
00079     connect(darts, SIGNAL(afficherNouvellePartie()), this, SLOT(
      afficherPartie()));
00080     connect(communication, SIGNAL(resetPartie()), this, SLOT(
      afficherNouvellePartie()));
00081     connect(darts, SIGNAL(finPartie(QString, int, bool)), this, SLOT(
      finirPartie(QString, int, bool)));
00082     connect(darts, SIGNAL(changementJoueurActif()), this, SLOT(
      mettreAJourJoueur()));
00083     connect(darts, SIGNAL(nouvelImpact(int, int, int)), this, SLOT(
      afficherImpact(int,int)));
00084     connect(darts, SIGNAL(miseAJourPoint()), this, SLOT(mettreAJourScore()));
00085     connect(darts, SIGNAL(miseAJourPointTournois()), this, SLOT(
      mettreAJourScoreTournois()));
00086     connect(darts, SIGNAL(nouvelleManche()), this, SLOT(mettreAJourManche()));
00087     connect(darts, SIGNAL(voleeAnnulee()), this, SLOT(afficherVoleeAnnulee()));
00088     connect(darts, SIGNAL(miseAJourMoyenneVolee()), this, SLOT(
      mettreAJourMoyenneVolee()));
00089     connect(darts, SIGNAL(miseAJourMoyenneVoleeTournois()), this, SLOT(
      mettreAJourMoyenneVoleeTournois()));
00090     connect(darts->getSolution(), SIGNAL(solutionTrouver(QString)), this, SLOT(
      mettreAJoursolution(QString)));
00091     connect(darts, SIGNAL(actualiserCible()), this, SLOT(mettreAJourCible()));
00092     connect(communication, SIGNAL(pause()), this, SLOT(
      mettrePausePartie()));
00093     connect(communication, SIGNAL(play()), this, SLOT(
      relancerpartie()));
00094     connect(communication, SIGNAL(erreurBluetooth(QString)), this, SLOT(
```

```
      mettreAJourMessageStatut(QString)));
00095     connect(darts, SIGNAL(jouerSon(QString)), this, SLOT(jouerSon(QString)));
00096     connect(communication, SIGNAL(jouerSon(QString)), this, SLOT(
      jouerSon(QString)));
00097     connect(communication, SIGNAL(afficherRegle(QString)), this, SLOT(
      lancerRegle(QString)));
00098     connect(communication, SIGNAL(stopperRegle()), this, SLOT(
      StopperLectureRegle()));
00099     connect(darts, SIGNAL(afficherTournois(QString, QString)), this, SLOT(
      initialiserAffichageTournois(QString, QString)));
00100     connect(darts, SIGNAL(debuterTournois()), this, SLOT(lancerTournois()));
00101     connect(darts, SIGNAL(changementJoueurActifTournoi()), this, SLOT(
      mettreAJourJoueurTournoi()));
00102     connect(darts, SIGNAL(finTournois(QString,QString,QList<Joueur>)), this, SLOT(
      afficherFinTournois(QString,QString,QList<Joueur>)));
00103     connect(darts, SIGNAL(afficherInfoTournois()), this, SLOT(
      afficherInformationTournois()));
00104
00105 }
```

7.3.4.22 initialiserHorloge()

```
void Ihm::initialiserHorloge ( )  [private]
```

initialise l'horloge pour un affichage périodique

Définition à la ligne 607 du fichier ihm.cpp.

Références actualiserHeure(), PERIODE_HORLOGE, et timerHorloge.

Référencé par Ihm().

```
00608 {
00609     timerHorloge = new QTimer(this); // Instancie dynamiquement le temporisateur du
       rafraichissement de l'heure
00610     connect(timerHorloge, SIGNAL(timeout()),this,SLOT(
      actualiserHeure())); // Pour le déclenchement périodique de l'affichage de l'heure
00611     timerHorloge->start(PERIODE_HORLOGE);  // Toutes les secondes (1000 ms)
00612 }
```

7.3.4.23 jouerSon

```
void Ihm::jouerSon (
            QString son )  [slot]
```

Méthode qui permet de jouer un son.

Paramètres

son	

Définition à la ligne 645 du fichier ihm.cpp.

Références CHEMIN_FICHIER_MUSIQUE.

Référencé par initialiserEvenements().

```
00646 {
00647     QSound::play(qApp->applicationDirPath() + CHEMIN_FICHIER_MUSIQUE + son);
```

```
00648     if(!QFileInfo(qApp->applicationDirPath() + CHEMIN_FICHIER_MUSIQUE + son).exists()
      )
00649     {
00650         qDebug() << Q_FUNC_INFO << "Pour avoir les sons, ajouter le pack disponible à cette adresse dans le
       build de votre application:"<<endl;
00651         qDebug() << Q_FUNC_INFO << "
      https://drive.google.com/file/d/1vH0tLe8lsu2VQLISDL94nJBA2arfLcbG/view?usp=sharing"<<endl;
00652     }
00653 }
```

7.3.4.24 lancerRegle

```
void Ihm::lancerRegle (
             QString regle )  [slot]
```

Méthode qui lance la vidéo explicative des regles suivant le type de jeu.

Paramètres

regle	

Définition à la ligne 678 du fichier ihm.cpp.

Références allerPage(), communication, Communication : :getEtatPartie(), Communication : :miseAJourEtatPartieRegle(), musique, musiquePause, PageRegle, player, et sauverEtatPartie.

Référencé par initialiserEvenements().

```
00679 {
00680     sauverEtatPartie = communication->getEtatPartie();
00681     communication->miseAJourEtatPartieRegle();
00682
00683     player->setMedia(QUrl::fromLocalFile(QCoreApplication::applicationDirPath() + QString("/" + regle
       + ".mp4")));
00684
00685     musiquePause.stop();
00686     musique.stop();
00687     allerPage(Ihm::PageRegle);
00688     player->play();
00689 }
```

7.3.4.25 lancerTournois

```
void Ihm::lancerTournois ( )  [slot]
```

methode qui affiche le l'ecran de tournois

Définition à la ligne 790 du fichier ihm.cpp.

Références afficherDureePartie(), allerPage(), compteurDureePartie, musique, PageTournois, et timerHorloge.

Référencé par initialiserEvenements().

```
00791 {
00792     musique.stop();
00793
00794     compteurDureePartie = 0;
00795     connect(timerHorloge, SIGNAL(timeout()),this,SLOT(
      afficherDureePartie())); // Pour le comptage et l'affichage de la durée d'une séance
00796
00797     allerPage(Ihm::PageTournois);
00798 }
```

7.3.4.26 mettreAJourCible

```
void Ihm::mettreAJourCible ( )  [slot]
```

Méthode qui reinitialise l'affichage de la cible.

Définition à la ligne 619 du fichier ihm.cpp.

Références messageStatut, sauvegardeImpactEncours, et ui.

Référencé par afficherNouvellePartie(), et initialiserEvenements().

```
00620 {
00621     ui->labelVisualisationimpact->setPixmap(QPixmap(":/ressources/cible.png"));
00622     ui->ImpactCibleTournois->setPixmap(QPixmap(":/ressources/cible.png"));
00623     sauvegardeImpactEncours = ui->labelVisualisationimpact->pixmap()->copy();
00624     messageStatut = "Volée ";
00625 }
```

7.3.4.27 mettreAJourJoueur

```
void Ihm::mettreAJourJoueur ( )  [slot]
```

Méthode qui initialise l'affichage du mode et des joueurs de la partie.

Définition à la ligne 258 du fichier ihm.cpp.

Références darts, Darts : :getJoueurActif(), Darts : :getListJoueur(), et ui.

Référencé par afficherPartie(), et initialiserEvenements().

```
00259 {
00260     QString nomjoueur;
00261
00262     int premierJoueurAfficher = 0;
00263     int dernierJoueurAfficher = darts->getListJoueur().size();
00264
00265     if(darts->getListJoueur().size() > 7)
00266     {
00267         premierJoueurAfficher = darts->getJoueurActif();
00268         dernierJoueurAfficher = darts->getJoueurActif() + 6;
00269
00270         while(dernierJoueurAfficher > darts->getListJoueur().size() ||
      premierJoueurAfficher < 0)
00271         {
00272             if(premierJoueurAfficher < 0)
00273             {
00274                 premierJoueurAfficher++;
00275                 dernierJoueurAfficher++;
00276             }
00277
00278             if(dernierJoueurAfficher > darts->getListJoueur().size())
00279             {
00280                 premierJoueurAfficher--;
00281                 dernierJoueurAfficher--;
00282             }
00283         }
00284     }
00285
00286     for(int i = premierJoueurAfficher; i < dernierJoueurAfficher; i++)
00287     {
00288         if(i == darts->getJoueurActif())    // test si le joueur est le joueur qui doit
       jouer
00289         {
00290             nomjoueur += "    " + darts->getListJoueur()[i].getNom() + "\n";  //joueur
       joue
00291         }
00292         else
00293         {
00294             nomjoueur += "            " + darts->getListJoueur()[i].getNom() + "\n";
      //joueur en attente de son tour //
00295         }
00296     }
00297     ui->nomJoueur->setText(nomjoueur);
00298 }
```

7.3.4.28 mettreAJourJoueurTournoi

```
void Ihm::mettreAJourJoueurTournoi ( )  [slot]
```

Méthode qui initialise l'affichage du mode et des joueurs de la partie.

Définition à la ligne 827 du fichier ihm.cpp.

Références darts, Darts : :getDernierJoueur(), Darts : :getJoueurActif(), Darts : :getListJoueur(), Darts : :getPremierJoueur(), et ui.

Référencé par initialiserEvenements().

```
00828 {
00829     if(darts->getJoueurActif() == darts->
      getPremierJoueur())
00830     {
00831         ui->nomJoueurTournois1->setText(" " + darts->getListJoueur()[
      darts->getPremierJoueur()].getNom() + " ");
00832         ui->nomJoueurTournois2->setText(darts->getListJoueur()[
      darts->getDernierJoueur()].getNom());
00833     }
00834     else
00835     {
00836         ui->nomJoueurTournois1->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom());
00837         ui->nomJoueurTournois2->setText(" " + darts->getListJoueur()[
      darts->getDernierJoueur()].getNom() + " ");
00838     }
00839 }
```

7.3.4.29 mettreAJourManche

```
void Ihm::mettreAJourManche ( )  [slot]
```

Méthode qui met à jour le numero de la manche.

Définition à la ligne 189 du fichier ihm.cpp.

Références darts, Darts : :getManche(), et ui.

Référencé par initialiserEvenements().

```
00190 {
00191     ui->manche->setText(QString::number(darts->getManche()));
00192     ui->tournoisManche->setText("Manche " + QString::number(darts->
      getManche()));
00193 }
```

7.3.4.30 mettreAJourMessageStatut() [1/2]

```
void Ihm::mettreAJourMessageStatut (
            int typePoint,
            int point )  [private]
```

Méthode qui met à jour le message de statut de la volée en cours.

Paramètres

typePoint	
point	

Définition à la ligne 230 du fichier ihm.cpp.

Références darts, DOUBLE_POINT, Darts : :getPointVolees(), messageStatut, TRIPLE_POINT, ui, et ZERO_POINT.

Référencé par afficherImpact(), et initialiserEvenements().

```
00231 {
00232
00233     switch(typePoint){
00234         case TRIPLE_POINT:
00235             messageStatut += " T" + QString::number(point);
00236         break;
00237         case DOUBLE_POINT:
00238             messageStatut += " D" + QString::number(point);
00239         break;
00240         case ZERO_POINT:
00241             messageStatut += " " + QString::number(ZERO_POINT);
00242         break;
00243         default:
00244             messageStatut += " " + QString::number(point);
00245         break;
00246     }
00247     ui->labelStatut->setStyleSheet("color: rgb(109, 43,107); border-color: rgb(109, 43,107);");
00248     ui->labelStatut->setText(messageStatut + "     " +QString::number(
      darts->getPointVolees()) + " Point(s)");
00249     ui->statutImpactTournois->setText(messageStatut + "     " +QString::number(
      darts->getPointVolees()) + " Point(s)");
00250 }
```

7.3.4.31 mettreAJourMessageStatut [2/2]

```
void Ihm::mettreAJourMessageStatut (
             QString statut )  [slot]
```

Méthode qui met à jour le message de statut.

Paramètres

statut	

Définition à la ligne 633 du fichier ihm.cpp.

Références ui.

```
00634 {
00635     ui->labelStatutAttente->setText(statut);
00636 }
```

7.3.4.32 mettreAJourMoyenneVolee

```
void Ihm::mettreAJourMoyenneVolee ( )  [slot]
```

Méthode qui initialise l'affichage du mode et des joueurs de la partie.

Définition à la ligne 305 du fichier ihm.cpp.

Références darts, Darts : :getListJoueur(), et ui.

Référencé par initialiserEvenements().

```
00306 {
00307     QString moyenneVoleeJoueur;
00308     QString moyenneVoleesJoueur;
00309
00310     int premierJoueurAfficher = 0;
00311     int dernierJoueurAfficher = darts->getListJoueur().size();
00312
00313     if(darts->getListJoueur().size() > 7)
00314     {
00315         for(int i = premierJoueurAfficher; i < 7; i++)
00316         {
00317             moyenneVoleeJoueur += darts->getListJoueur()[i].getNom() + " : " +
      QString::number(darts->getListJoueur()[i].getMoyenneVolee()) +" \n"; //"          " +
00318         }
00319
00320         for(int i = 7; i < dernierJoueurAfficher; i++)
00321         {
00322             moyenneVoleesJoueur += darts->getListJoueur()[i].getNom() + " : " +
      QString::number(darts->getListJoueur()[i].getMoyenneVolee()) +" \n"; //"          " +
00323         }
00324
00325         ui->moyenneVolee->setText(moyenneVoleeJoueur);
00326         ui->lineSeparateurMoyenne->setVisible(true);
00327         ui->moyenneVolee2->setText(moyenneVoleesJoueur);
00328     }
00329     else
00330     {
00331         for(int i = premierJoueurAfficher; i < dernierJoueurAfficher; i++)
00332         {
00333             moyenneVoleeJoueur += darts->getListJoueur()[i].getNom() + " : " +
      QString::number(darts->getListJoueur()[i].getMoyenneVolee()) +" \n"; //"          " +
00334         }
00335         ui->moyenneVolee->setText(moyenneVoleeJoueur);
00336     }
00337     ui->labelMoyenneVolees->setVisible(true);
00338     ui->lineScoreActuel->setVisible(true);
00339     ui->labelMoyenneVoleesStatistique->setVisible(true);
00340     ui->moyenneVolees->setText(moyenneVoleeJoueur + moyenneVoleesJoueur);
00341 }
```

7.3.4.33 mettreAJourMoyenneVoleeTournois

```
void Ihm::mettreAJourMoyenneVoleeTournois ( )  [slot]
```

Méthode qui met a jour la moyenne des Volee du joueur.

Définition à la ligne 816 du fichier ihm.cpp.

Références darts, Darts : :getDernierJoueur(), Darts : :getListJoueur(), Darts : :getPremierJoueur(), et ui.

Référencé par initialiserEvenements().

```
00817 {
00818     ui->moyenneJoueurTournois1->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom() + "\n\n" + QString::number(
      darts->getListJoueur()[darts->getPremierJoueur()].getMoyenneVolee())
      );
00819     ui->moyenneJoueurTournois2->setText(darts->getListJoueur()[
      darts->getDernierJoueur()].getNom() + "\n\n" +QString::number(
      darts->getListJoueur()[darts->getDernierJoueur()].getMoyenneVolee())
      );
00820 }
```

7.3.4.34 mettreAJourScore

```
void Ihm::mettreAJourScore ( )  [slot]
```

Méthode qui met à jour le score dans L'ihm.

Définition à la ligne 151 du fichier ihm.cpp.

Références darts, Darts : :getJoueurActif(), Darts : :getListJoueur(), et ui.

Référencé par afficherPartie(), et initialiserEvenements().

```
00152 {
00153     QString score;
00154     int premierJoueurAfficher = 0;
00155     int dernierJoueurAfficher = darts->getListJoueur().size();
00156 
00157     if(darts->getListJoueur().size() > 7)
00158     {
00159         premierJoueurAfficher = darts->getJoueurActif();
00160         dernierJoueurAfficher = darts->getJoueurActif() + 6;
00161 
00162         while(dernierJoueurAfficher > darts->getListJoueur().size() ||
      premierJoueurAfficher < 0)
00163         {
00164             if(premierJoueurAfficher < 0)
00165             {
00166                 premierJoueurAfficher++;
00167                 dernierJoueurAfficher++;
00168             }
00169 
00170             if(dernierJoueurAfficher > darts->getListJoueur().size())
00171             {
00172                 premierJoueurAfficher--;
00173                 dernierJoueurAfficher--;
00174             }
00175         }
00176     }
00177     for(int i = premierJoueurAfficher; i < dernierJoueurAfficher; i++)
00178     {
00179         score += darts->getListJoueur()[i].getNom() + " : " + QString::number(
      darts->getListJoueur()[i].getScore()) + "\n"; // "          " +
00180     }
00181     ui->scoreActuel->setText(score);
00182 }
```

7.3.4.35 mettreAJourScoreTournois

```
void Ihm::mettreAJourScoreTournois ( )  [slot]
```

methode qui met a jour le score des joueurs des tournois

Définition à la ligne 805 du fichier ihm.cpp.

Références darts, Darts : :getDernierJoueur(), Darts : :getListJoueur(), Darts : :getPremierJoueur(), et ui.

Référencé par initialiserEvenements().

```
00806 {
00807     ui->scoreJoueurTournois1->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom() + "\n\n" + QString::number(
      darts->getListJoueur()[darts->getPremierJoueur()].getScore()));
00808     ui->scoreJoueurTournois2->setText(darts->getListJoueur()[
      darts->getDernierJoueur()].getNom() + "\n\n" + QString::number(
      darts->getListJoueur()[darts->getDernierJoueur()].getScore()));
00809 }
```

7.3.4.36 mettreAJoursolution

```
void Ihm::mettreAJoursolution (
            QString solution )  [slot]
```

Affiche les solutions possibles pour finir la parties.

Paramètres

solution	

Définition à la ligne 559 du fichier ihm.cpp.

Références ui.

Référencé par initialiserEvenements().

```
00560 {
00561     ui->labelStatut->setStyleSheet("color: rgb(179, 0,5); border-color: rgb(179, 0,5);");
00562     ui->labelStatut->setText(solution);
00563     ui->statutImpactTournois->setText(solution);
00564 }
```

7.3.4.37 mettrePausePartie

```
void Ihm::mettrePausePartie ( )  [slot]
```

Mets en pause le chronométrage de la partiee.

Définition à la ligne 571 du fichier ihm.cpp.

Références afficherDureePartie(), musiquePause, sauvegardeImpactEncours, timerHorloge, et ui.

Référencé par initialiserEvenements().

```
00572 {
00573     disconnect(timerHorloge, SIGNAL(timeout()),this,SLOT(
      afficherDureePartie())); // mettre en pause le chronometrage de la partie
00574     sauvegardeImpactEncours = ui->labelVisualisationimpact->pixmap()->copy();
00575     QImage pause(":pause.png");
00576     QPixmap cibleImpacte = ui->labelVisualisationimpact->pixmap()->copy(); // on récupère l'image
       précédente;
00577     QPainter p(&cibleImpacte);
00578     p.drawImage(QPoint(0, 0), pause);
00579     p.end();
00580     ui->labelVisualisationimpact->setPixmap(cibleImpacte);
00581     ui->ImpactCibleTournois->setPixmap(cibleImpacte);
00582     ui->labelTempsPartie->setStyleSheet("color: rgb(179, 0,5);");
00583
00584     musiquePause.play();
00585 }
```

7.3.4.38 relancerpartie

```
void Ihm::relancerpartie ( )  [slot]
```

relancer le chronométrage de la partie

Définition à la ligne 592 du fichier ihm.cpp.

Références afficherDureePartie(), musiquePause, sauvegardeImpactEncours, timerHorloge, et ui.

Référencé par initialiserEvenements(), et testerEtatPartie().

```
00593 {
00594     ui->labelVisualisationimpact->setPixmap(sauvegardeImpactEncours);
00595     ui->ImpactCibleTournois->setPixmap(sauvegardeImpactEncours);
00596     ui->labelTempsPartie->setStyleSheet("color: rgb(109, 43,107);");
00597     connect(timerHorloge, SIGNAL(timeout()),this,SLOT(
      afficherDureePartie())); // relancer le chronometrage de la partie
00598     qDebug() << Q_FUNC_INFO << endl;
00599     musiquePause.stop();
00600 }
```

7.3.4.39 stateChanged

```
void Ihm::stateChanged (
            QMediaPlayer::State state )  [slot]
```

Méthode appelée quand l'état de la vidéo change.

Paramètres

state	

Définition à la ligne 730 du fichier ihm.cpp.

Références testerEtatPartie().

Référencé par initialiserAffichageRegle().

```
00731 {
00732     qDebug() << Q_FUNC_INFO << state;
00733     if(state == QMediaPlayer::StoppedState)
00734     {
00735         testerEtatPartie();
00736     }
00737 }
```

7.3.4.40 StopperLectureRegle

```
void Ihm::StopperLectureRegle ( )  [slot]
```

methode qui stop la lecture de la musique

Définition à la ligne 744 du fichier ihm.cpp.

Références player.

Référencé par initialiserEvenements().

```
00745 {
00746     player->stop();
00747 }
```

7.3.4.41 testerEtatPartie()

```
void Ihm::testerEtatPartie ( )  [private]
```

Méthode appelée pour remettre l'état dans lequel était la partie avant l'affichage des règles.

Définition à la ligne 696 du fichier ihm.cpp.

Références allerPage(), communication, Communication : :miseAJourEtatPartieAttente(), Communication : :miseAJourEtatPartie↩EnCours(), Communication : :miseAJourEtatPartieFin(), Communication : :miseAJourEtatPartiePause(), musique, musiquePause, PageAttente, PageJeu, PageStatistique, relancerpartie(), et sauverEtatPartie.

Référencé par error(), et stateChanged().

```
00697 {
00698     if(sauverEtatPartie == 0)
00699     {
00700         communication->miseAJourEtatPartieAttente();
00701         musique.play();
00702         allerPage(Ihm::PageAttente);
00703     }
00704     else if(sauverEtatPartie == 1)
00705     {
00706         communication->miseAJourEtatPartieEnCours();
00707         allerPage(Ihm::PageJeu);
00708         relancerpartie();
00709     }
00710     else if(sauverEtatPartie == 2)
00711     {
00712         communication->miseAJourEtatPartieFin();
00713         musique.play();
00714         allerPage(Ihm::PageStatistique);
00715     }
00716     else if(sauverEtatPartie == 3)
00717     {
00718         communication->miseAJourEtatPartiePause();
00719         musiquePause.play();
00720         allerPage(Ihm::PageJeu);
00721     }
00722 }
```

7.3.5 Documentation des données membres

7.3.5.1 communication

```
Communication* Ihm::communication  [private]
```

objet communication

Définition à la ligne 54 du fichier ihm.h.

Référencé par Ihm(), initialiserEvenements(), lancerRegle(), et testerEtatPartie().

7.3.5.2 compteurDureePartie

```
int Ihm::compteurDureePartie  [private]
```

compteur de secondes pour la durée d'une séance

Définition à la ligne 58 du fichier ihm.h.

Référencé par afficherDureePartie(), afficherPartie(), et lancerTournois().

7.3.5.3 darts

```
Darts* Ihm::darts  [private]
```

objet darts

Définition à la ligne 55 du fichier ihm.h.

Référencé par afficherPartie(), finirPartie(), Ihm(), initialiserAffichageTournois(), initialiserEvenements(), mettreAJourJoueur(), mettreAJourJoueurTournoi(), mettreAJourManche(), mettreAJourMessageStatut(), mettreAJourMoyenneVolee(), mettreAJourMoyenneVoleeTournois(), mettreAJourScore(), et mettreAJourScoreTournois().

7.3.5.4 messageStatut

```
QString Ihm::messageStatut  [private]
```

contient le message de statut qui est affiché

Définition à la ligne 61 du fichier ihm.h.

Référencé par mettreAJourCible(), et mettreAJourMessageStatut().

7.3.5.5 musique

```
QSound Ihm::musique  [private]
```

objet musique

Définition à la ligne 56 du fichier ihm.h.

Référencé par afficherNouvellePartie(), afficherPartie(), finirPartie(), lancerRegle(), lancerTournois(), et testerEtatPartie().

7.3.5.6 musiquePause

```
QSound Ihm::musiquePause  [private]
```

objet musiquePause

Définition à la ligne 57 du fichier ihm.h.

Référencé par afficherNouvellePartie(), lancerRegle(), mettrePausePartie(), relancerpartie(), et testerEtatPartie().

7.3.5.7 player

```
QMediaPlayer* Ihm::player  [private]
```

objet player

Définition à la ligne 64 du fichier ihm.h.

Référencé par afficherNouvellePartie(), error(), finirPartie(), initialiserAffichageRegle(), lancerRegle(), et StopperLectureRegle().

7.3.5.8 sauvegardeImpactEncours

```
QPixmap Ihm::sauvegardeImpactEncours  [private]
```

sauvegarde le QPixmap de l'état de la cible

Définition à la ligne 60 du fichier ihm.h.

Référencé par mettreAJourCible(), mettrePausePartie(), et relancerpartie().

7.3.5.9 sauverEtatPartie

```
int Ihm::sauverEtatPartie  [private]
```

Contient l'état de la partie avant l'affichage des règles.

Définition à la ligne 59 du fichier ihm.h.

Référencé par lancerRegle(), et testerEtatPartie().

7.3.5.10 timerHorloge

```
QTimer* Ihm::timerHorloge  [private]
```

objet timerHorloge

Définition à la ligne 53 du fichier ihm.h.

Référencé par afficherPartie(), finirPartie(), initialiserHorloge(), lancerTournois(), mettrePausePartie(), et relancerpartie().

7.3.5.11 ui

```
Ui::Ihm* Ihm::ui  [private]
```

object de notre Ihm

Définition à la ligne 52 du fichier ihm.h.

Référencé par actualiserHeure(), afficherAttenteConfiguration(), afficherAttenteConnexion(), afficherDureePartie(), afficherFin↩Tournois(), afficherImpact(), afficherInformationTournois(), afficherNouvellePartie(), afficherPartie(), afficherPretLancerTournois(), afficherVoleeAnnulee(), allerPage(), allerPagePrecedente(), allerPageSuivante(), finirPartie(), Ihm(), initialiserAffichageRegle(), initialiserAffichageTournois(), mettreAJourCible(), mettreAJourJoueur(), mettreAJourJoueurTournoi(), mettreAJourManche(), mettre↩AJourMessageStatut(), mettreAJourMoyenneVolee(), mettreAJourMoyenneVoleeTournois(), mettreAJourScore(), mettreAJourScore↩Tournois(), mettreAJoursolution(), mettrePausePartie(), relancerpartie(), et ∼Ihm().

7.3.5.12 videoWidget

```
QVideoWidget* Ihm::videoWidget  [private]
```

objet videoWidget

Définition à la ligne 65 du fichier ihm.h.

Référencé par initialiserAffichageRegle().

La documentation de cette classe a été générée à partir des fichiers suivants :

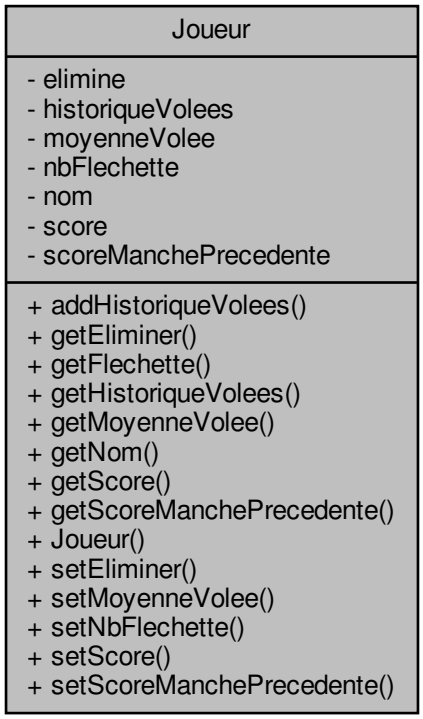
— ihm.h
— ihm.cpp

7.4 Référence de la classe Joueur

Déclaration de la classe Joueur (Module Ecran-DARTS)

```
#include <joueur.h>
```

Graphe de collaboration de Joueur :



Fonctions membres publiques

- void addHistoriqueVolees (float volee)
  Méthode qui ajoute la volée au vecteur contenant l'historique des volées.
- bool getEliminer () const
  Retourne un etat true/false pour savoir si le joueur est eliminé
- int getFlechette () const
  Retourne le nombre de flechette du joueur.
- QVector< float > getHistoriqueVolees () const
  Retourne le vector contenant tous les scores des volées precedente.
- int getMoyenneVolee () const
  Retourne la moyenne des volees.
- QString getNom () const
  Retourne le nom du joueur.
- int getScore () const
  Retourne le score du joueur.
- int getScoreManchePrecedente () const
  Retourne le score de la manche precedente.
- Joueur (QString nom, int score, int nbFlechette)
  Constructeur de la classe Joueur.
- void setEliminer (bool elimine)
  permet de modifier si le joueur est eliminer
- void setMoyenneVolee (int moyenneVolee)

Permets de mettre à jour la moyenne des volées.
- void setNbFlechette (int nbFlechette)
  Permets de mettre à jour le nombre de fléchette du joueur.
- void setScore (int score)
  Permets de mettre à jour le score du joueur.
- void setScoreManchePrecedente (int scoreManchePrecedente)
  Permets de mettre à jour le score de la manche précédente.

Attributs privés

- bool elimine
  contient un état true/false pour savoir si le joueur est éliminé.
- QVector< float > historiqueVolees
  contient l'historique des volées du joueur
- int moyenneVolee
  contient la moyenne des volées du joueur
- int nbFlechette
  contient le nombre de flechette restante au joueur
- QString nom
  contient le nom du joueur
- int score
  contient le score du joueur
- int scoreManchePrecedente
  contient le score de la manche precedente

7.4.1 Description détaillée

Déclaration de la classe Joueur (Module Ecran-DARTS)

Cette classe s'occupe du stockage du nom, score et nombre de fléchettes du joueur

Définition à la ligne 21 du fichier joueur.h.

7.4.2 Documentation des constructeurs et destructeur

7.4.2.1 Joueur()

```
Joueur::Joueur (
            QString nom,
            int score,
            int nbFlechette )
```

Constructeur de la classe Joueur.

Paramètres

nom	
score	
nbFlechette	

Définition à la ligne 24 du fichier joueur.cpp.

Références nbFlechette.

```
00024                                                   : nom(nom), score(
      score), moyenneVolee(0), scoreManchePrecedente(
      score), nbFlechette(nbFlechette), elimine(false)
00025 {
00026     qDebug() << Q_FUNC_INFO << nom << " " << score << " " << nbFlechette;
00027 }
```

7.4.3 Documentation des fonctions membres

7.4.3.1 addHistoriqueVolees()

```
void Joueur::addHistoriqueVolees (
              float volee )
```

Méthode qui ajoute la volée au vecteur contenant l'historique des volées.

Paramètres

volee	

Définition à la ligne 168 du fichier joueur.cpp.

Références historiqueVolees.

```
00169 {
00170     historiqueVolees.push_back(volee);
00171 }
```

7.4.3.2 getEliminer()

```
bool Joueur::getEliminer ( ) const
```

Retourne un etat true/false pour savoir si le joueur est eliminé

Renvoie

bool

Définition à la ligne 101 du fichier joueur.cpp.

Références elimine.

```
00102 {
00103     return this->elimine;
00104 }
```

7.4.3.3 getFlechette()

```
int Joueur::getFlechette ( ) const
```

Retourne le nombre de flechette du joueur.

Renvoie

int

Définition à la ligne 68 du fichier joueur.cpp.

Références nbFlechette.

```
00069 {
00070     return this->nbFlechette;
00071 }
```

7.4.3.4 getHistoriqueVolees()

```
QVector< float > Joueur::getHistoriqueVolees ( ) const
```

Retourne le vector contenant tous les scores des volées precedente.

Renvoie

QVector<float>

Définition à la ligne 90 du fichier joueur.cpp.

Références historiqueVolees.

```
00091 {
00092     return this->historiqueVolees;
00093 }
```

7.4.3.5 getMoyenneVolee()

```
int Joueur::getMoyenneVolee ( ) const
```

Retourne la moyenne des volees.

Renvoie

float

Définition à la ligne 79 du fichier joueur.cpp.

Références moyenneVolee.

```
00080 {
00081     return this->moyenneVolee;
00082 }
```

7.4.3.6 getNom()

```
QString Joueur::getNom ( ) const
```

Retourne le nom du joueur.

Renvoie

QString

Définition à la ligne 35 du fichier joueur.cpp.

Références nom.

```
00036 {
00037     return this->nom;
00038 }
```

7.4.3.7 getScore()

```
int Joueur::getScore ( ) const
```

Retourne le score du joueur.

Renvoie

int

Définition à la ligne 46 du fichier joueur.cpp.

Références score.

```
00047 {
00048     return this->score;
00049 }
```

7.4.3.8 getScoreManchePrecedente()

```
int Joueur::getScoreManchePrecedente ( ) const
```

Retourne le score de la manche precedente.

Renvoie

int

Définition à la ligne 57 du fichier joueur.cpp.

Références scoreManchePrecedente.

```
00058 {
00059     return this->scoreManchePrecedente;
00060 }
```

7.4.3.9 setEliminer()

```
void Joueur::setEliminer (
            bool elimine )
```

permet de modifier si le joueur est eliminer

Paramètres

elimine	

Définition à la ligne 112 du fichier joueur.cpp.

Références elimine.

```
00113 {
00114     this->elimine = elimine;
00115 }
```

7.4.3.10 setMoyenneVolee()

```
void Joueur::setMoyenneVolee (
            int moyenneVolee )
```

Permets de mettre à jour la moyenne des volées.

Paramètres

moyenneVolee	

Définition à la ligne 123 du fichier joueur.cpp.

Références moyenneVolee.

```
00124 {
00125     this->moyenneVolee = moyenneVolee;
00126 }
```

7.4.3.11 setNbFlechette()

```
void Joueur::setNbFlechette (
            int nbFlechette )
```

Permets de mettre à jour le nombre de fléchette du joueur.

Paramètres

nbFlechette	

Définition à la ligne 157 du fichier joueur.cpp.

Références nbFlechette.

```
00158 {
00159     this->nbFlechette = nbFlechette;
00160 }
```

7.4.3.12 setScore()

```
void Joueur::setScore (
            int score )
```

Permets de mettre à jour le score du joueur.

Paramètres

score	

Définition à la ligne 135 du fichier joueur.cpp.

Références score.

```
00136 {
00137     this->score = score;
00138 }
```

7.4.3.13 setScoreManchePrecedente()

```
void Joueur::setScoreManchePrecedente (
            int scoreManchePrecedente )
```

Permets de mettre à jour le score de la manche précédente.

Paramètres

scoreManchePrecedente	

Définition à la ligne 146 du fichier joueur.cpp.

Références scoreManchePrecedente.

```
00147 {
00148     this->scoreManchePrecedente = scoreManchePrecedente;
00149 }
```

7.4.4 Documentation des données membres

7.4.4.1 elimine

```
bool Joueur::elimine  [private]
```

contient un état true/false pour savoir si le joueur est éliminé.

Définition à la ligne 47 du fichier joueur.h.

Référencé par getEliminer(), et setEliminer().

7.4.4.2 historiqueVolees

```
QVector<float> Joueur::historiqueVolees  [private]
```

contient l'historique des volées du joueur

Définition à la ligne 42 du fichier joueur.h.

Référencé par addHistoriqueVolees(), et getHistoriqueVolees().

7.4.4.3 moyenneVolee

```
int Joueur::moyenneVolee  [private]
```

contient la moyenne des volées du joueur

Définition à la ligne 44 du fichier joueur.h.

Référencé par getMoyenneVolee(), et setMoyenneVolee().

7.4.4.4 nbFlechette

```
int Joueur::nbFlechette  [private]
```

contient le nombre de flechette restante au joueur

Définition à la ligne 46 du fichier joueur.h.

Référencé par getFlechette(), Joueur(), et setNbFlechette().

7.4.4.5 nom

```
QString Joueur::nom  [private]
```

contient le nom du joueur

Définition à la ligne 41 du fichier joueur.h.

Référencé par getNom().

7.4.4.6 score

```
int Joueur::score  [private]
```

contient le score du joueur

Définition à la ligne 43 du fichier joueur.h.

Référencé par getScore(), et setScore().

7.4.4.7 scoreManchePrecedente

```
int Joueur::scoreManchePrecedente  [private]
```

contient le score de la manche precedente

Définition à la ligne 45 du fichier joueur.h.

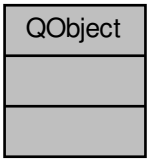
Référencé par getScoreManchePrecedente(), et setScoreManchePrecedente().

La documentation de cette classe a été générée à partir des fichiers suivants :
- joueur.h
- joueur.cpp

7.5 Référence de la classe QObject

La classe QObject est la classe de base de tous les objets Qt. Elle permet à ces objets Qt de disposer entre autres du mécanisme de communication signal/slot.

Graphe de collaboration de QObject :



7.5.1 Description détaillée

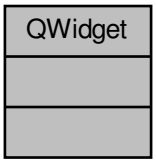
La classe QObject est la classe de base de tous les objets Qt. Elle permet à ces objets Qt de disposer entre autres du mécanisme de communication signal/slot.

La documentation de cette classe a été générée à partir du fichier suivant :
- main.cpp

7.6 Référence de la classe QWidget

La classe QWidget est la classe de base de tous les objets graphiques d'interface utilisateur.

Graphe de collaboration de QWidget :



7.6.1 Description détaillée

La classe QWidget est la classe de base de tous les objets graphiques d'interface utilisateur.

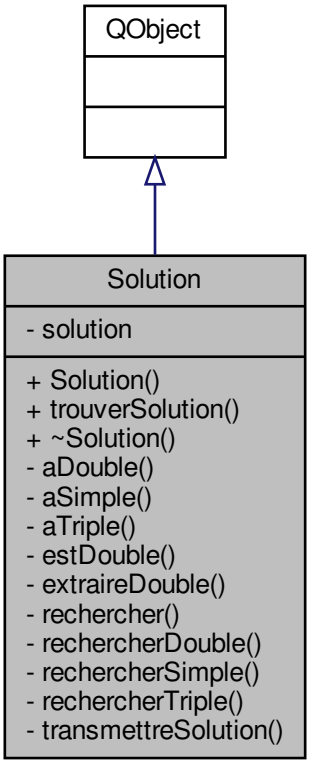
La documentation de cette classe a été générée à partir du fichier suivant :
— main.cpp

7.7 Référence de la classe Solution

Déclaration de la classe Solution (Module Ecran-DARTS)

```
#include <solution.h>
```

Graphe de collaboration de Solution :



Signaux

— void solutionTrouver (QString solution)

Fonctions membres publiques

— Solution (QObject ∗parent=nullptr)
Constructeur de la classe Solution.
— void trouverSolution (int s, int flechettes)
Méthode qui trouve la meilleure solution.
— ∼Solution ()
Destructeur de la classe Solution.

Fonctions membres privées

- bool aDouble (int points, const int score)
  Test si le double choisie et possible.
- bool aSimple (int points, const int score)
  Test si le simple choisi et possible.
- bool aTriple (int points, const int score)
  Test si la triple choisie et possible.
- bool estDouble (int points, const int score)
  Méthode qui teste si les points son double.
- bool extraireDouble (int &score, int cible)
  Méthode qui cherche le double pour finir la partie.
- bool rechercher (int score, int nbFlechettes, bool still=false)
  Méthode qui recherche la meilleure combinaison pour finir.
- bool rechercherDouble (int &score, QString &combinaison)
  Méthode qui recherche le meilleur double pour la solution.
- bool rechercherSimple (int &score, QString &combinaison)
  Méthode qui recherche le meilleur simple pour la solution.
- bool rechercherTriple (int &score, QString &combinaison, int start)
  Méthode qui recherche le meilleur triple pour la solution.
- void transmettreSolution (int score)
  Méthode qui émet un signal pour que l'Ihm affiche la solution trouver.

Attributs privés

- QString solution
  contient la solution pour finir la partie

7.7.1 Description détaillée

Déclaration de la classe Solution (Module Ecran-DARTS)

Cette classe s'occupe de rechercher les différentes solutions pour finir la partie

Définition à la ligne 30 du fichier solution.h.

7.7.2 Documentation des constructeurs et destructeur

7.7.2.1 Solution()

```
Solution::Solution (
            QObject * parent = nullptr )  [explicit]
```

Constructeur de la classe Solution.

Paramètres

parent	

Définition à la ligne 20 du fichier solution.cpp.

```
00020                                   : QObject(parent), solution("")
00021 {
00022     qDebug() << Q_FUNC_INFO;
00023 }
```

7.7.2.2 ~Solution()

```
Solution::~Solution ( )
```

Destructeur de la classe Solution.

Définition à la ligne 30 du fichier solution.cpp.

```
00031 {
00032     qDebug() << Q_FUNC_INFO;
00033 }
```

7.7.3 Documentation des fonctions membres

7.7.3.1 aDouble()

```
bool Solution::aDouble (
            int points,
            const int score )  [private]
```

Test si le double choisie et possible.

Paramètres

points	
score	

Renvoie

bool

Définition à la ligne 97 du fichier solution.cpp.

Référencé par extraireDouble(), et rechercherDouble().

```
00098 {
00099     if((score / (points*2)) >= 1)
00100     {
00101         return true;
00102     }
00103     return false;
00104 }
```

7.7.3.2 aSimple()

```
bool Solution::aSimple (
            int points,
            const int score )  [private]
```

Test si le simple choisi et possible.

Paramètres

points	
score	

Renvoie

bool

Définition à la ligne 141 du fichier solution.cpp.

Référencé par rechercherSimple().

```
00142 {
00143     if((score / points) >= 1)
00144         return true;
00145     return false;
00146 }
```

7.7.3.3 aTriple()

```
bool Solution::aTriple (
            int points,
            const int score )  [private]
```

Test si la triple choisie et possible.

Paramètres

points	
score	

Renvoie

bool

Définition à la ligne 56 du fichier solution.cpp.

Référencé par rechercherTriple().

```
00057 {
00058     if((score / (points*3)) >= 1)
00059     {
00060         return true;
00061     }
00062     return false;
00063 }
```

7.7.3.4 estDouble()

```
bool Solution::estDouble (
            int points,
            const int score )  [private]
```

Méthode qui teste si les points son double.

Paramètres

points	
score	

Renvoie

bool

Définition à la ligne 183 du fichier solution.cpp.

```
00184 {
00185     if(score == (points*2))
00186         return true;
00187     return false;
00188 }
```

7.7.3.5 extraireDouble()

```
bool Solution::extraireDouble (
            int & score,
            int cible )  [private]
```

Méthode qui cherche le double pour finir la partie.

Paramètres

score	
cible	

Renvoie

bool

Définition à la ligne 198 du fichier solution.cpp.

Références aDouble().

Référencé par trouverSolution().

```
00199 {
00200     if(aDouble(cible, score))
00201     {
00202         score -= (cible*2);
00203         return true;
00204     }
00205     return false;
00206 }
```

7.7.3.6 rechercher()

```
bool Solution::rechercher (
            int score,
            int nbFlechettes,
            bool still = false )  [private]
```

Méthode qui recherche la meilleure combinaison pour finir.

Paramètres

score	
nbFlechettes	
still	

Renvoie

bool

Définition à la ligne 217 du fichier solution.cpp.

Références RECHERCHE_DOUBLE, RECHERCHE_FINIE, RECHERCHE_SIMPLE, RECHERCHE_TRIPLE, rechercherDouble(), rechercherSimple(), rechercherTriple(), et solution.

Référencé par trouverSolution().

```
00218 {
00219     QString combinaison;
00220     int s = score;
00221     int n = RECHERCHE_TRIPLE;
00222     int start = 20;
00223     bool trouve = false;
00224
00225     solution.clear();
00226     for(int flechettes=nbFlechettes; flechettes>0 && score>0;)
00227     {
00228         if(n == RECHERCHE_TRIPLE)
00229         {
00230             trouve = rechercherTriple(score, combinaison, start);
00231             if(!trouve)
00232             {
00233                 n = RECHERCHE_DOUBLE;
00234             }
00235         }
00236         else if(n == RECHERCHE_DOUBLE)
00237         {
00238             trouve = rechercherDouble(score, combinaison);
00239             if(!trouve)
00240             {
00241                 n = RECHERCHE_SIMPLE;
00242             }
00243         }
00244         else if(n == RECHERCHE_SIMPLE)
00245         {
00246             trouve = rechercherSimple(score, combinaison);
00247         }
00248
00249         if(trouve)
00250         {
00251             solution +=  " " + combinaison;
00252             flechettes--;
00253             if(score == 0)
00254             {
00255                 return true;
00256             }
00257         }
00258
00259         // pas trouvé ?
00260         if(flechettes == 0 && score != 0)
00261         {
00262             if(still)
00263                 return true;
00264             // on recommence
00265             solution.clear();
00266             score = s;
00267             flechettes = nbFlechettes;
00268             n++; // on change de tactique
00269         }
00270
00271         if(n == RECHERCHE_FINIE)
00272         {
00273             start--; // on essaye un autre triple
00274             // on recommence
00275             solution.clear();
00276             n = RECHERCHE_TRIPLE;
00277             score = s;
00278             flechettes = nbFlechettes;
00279         }
00280
```

```
00281         if(start == 1)
00282         {
00283             return false;
00284         }
00285     }
00286     return false;
00287 }
```

7.7.3.7 rechercherDouble()

```
bool Solution::rechercherDouble (
            int & score,
            QString & combinaison )  [private]
```

Méthode qui recherche le meilleur double pour la solution.

Paramètres

score	
combinaison	

Renvoie

bool

Définition à la ligne 114 du fichier solution.cpp.

Références aDouble(), et BULL.

Référencé par rechercher().

```
00115 {
00116     bool trouve = false;
00117     // Remarque : le 2 est plus facile que le D1 !
00118     for(int i=BULL; i>1 && !trouve; i--)
00119     {
00120         // cibles inexistantes ?
00121         if(i > 20)
00122             continue;
00123         if(aDouble(i, score))
00124         {
00125             combinaison = "D" + QString::number(i);
00126             score -= (i*2);
00127             trouve = true;
00128         }
00129     }
00130     return trouve;
00131 }
```

7.7.3.8 rechercherSimple()

```
bool Solution::rechercherSimple (
            int & score,
            QString & combinaison )  [private]
```

Méthode qui recherche le meilleur simple pour la solution.

Paramètres

score	
combinaison	

Renvoie

bool

Définition à la ligne 156 du fichier solution.cpp.

Références aSimple(), et BULL.

Référencé par rechercher().

```
00157 {
00158     bool trouve = false;
00159
00160     for(int i=BULL; i>0 && !trouve; i--)
00161     {
00162         // cibles inexistantes ?
00163         if(i > 20)
00164             continue;
00165         if(aSimple(i, score))
00166         {
00167             combinaison = QString::number(i);
00168             score -= i;
00169             trouve = true;
00170         }
00171     }
00172     return trouve;
00173 }
```

7.7.3.9 rechercherTriple()

```
bool Solution::rechercherTriple (
            int & score,
            QString & combinaison,
            int start = 20 )  [private]
```

Méthode qui recherche le meilleur triple pour la solution.

Paramètres

score	
combinaison	
start	

Renvoie

bool

Définition à la ligne 74 du fichier solution.cpp.

Références aTriple().

Référencé par rechercher().

```
00075 {
00076     bool trouve = false;
00077     for(int i=start; i>1 && !trouve; i--)
00078     {
00079         if(aTriple(i, score))
00080         {
00081             combinaison = "T" + QString::number(i);
00082             score -= (i*3);
00083             trouve = true;
00084         }
00085     }
00086     return trouve;
00087 }
```

7.7.3.10 solutionTrouver

```
void Solution::solutionTrouver (
            QString solution )  [signal]
```

Référencé par transmettreSolution().

7.7.3.11 transmettreSolution()

```
void Solution::transmettreSolution (
            int score )  [private]
```

Méthode qui émet un signal pour que l'Ihm affiche la solution trouver.

Paramètres

score	

Définition à la ligne 42 du fichier solution.cpp.

Références solution, et solutionTrouver().

Référencé par trouverSolution().

```
00043 {
00044     qDebug() << Q_FUNC_INFO << "Score = " << score << " : " << solution;
00045     emit solutionTrouver("  " + QString::number(score) + "  " + solution + "  ");
00046 }
```

7.7.3.12 trouverSolution()

```
void Solution::trouverSolution (
            int s,
            int flechettes )
```

Méthode qui trouve la meilleure solution.

Paramètres

s	
flechettes	

Définition à la ligne 296 du fichier solution.cpp.

Références BULL, extraireDouble(), rechercher(), solution, et transmettreSolution().

Référencé par Darts : :configurationTournois(), Darts : :enleverPointImpact(), Darts : :gererManche(), Darts : :gererMancheTournois(), et Darts : :initialiserPartie().

```
00297 {
00298     int score = 0;
00299     bool trouve = false;
00300     int nbFlechettes = 0;
00301     solution = "";
00302 
00303     trouve = false;
00304     for(int i=BULL; i>0 && !trouve; i--)
00305     {
00306         // cibles inexistantes ?
00307         if(i > 20)
00308             continue;
00309         score = scoreJoueur; // <- le score à determiner
00310         nbFlechettes = flechettes; // <- le nombre de fléchettes
00311         if(extraireDouble(score, i))
00312         {
00313             nbFlechettes--;
00314             if(rechercher(score, nbFlechettes) || score == 0)
00315             {
00316                 solution += " D" + QString::number(i) + "*";
00317                 transmettreSolution(scoreJoueur);
00318                 trouve = true;
00319                 break;
00320             }
00321         }
00322     }
00323     if(!trouve)
00324     {
00325         rechercher(score, nbFlechettes+1, true);
00326         //transmettreSolution(scoreJoueur);       //activer pour avoir l'aide tout le temps même quand on
      ne peut pas finir
00327     }
00328 }
```

7.7.4 Documentation des données membres

7.7.4.1 solution

```
QString Solution::solution  [private]
```

contient la solution pour finir la partie

Définition à la ligne 39 du fichier solution.h.

Référencé par rechercher(), transmettreSolution(), et trouverSolution().

La documentation de cette classe a été générée à partir des fichiers suivants :
— solution.h
— solution.cpp

8 Documentation des fichiers

8.1 Référence du fichier Changelog.md

8.2 Changelog.md

```
00001 \page page_changelog Changelog
00002 
00003 r1 | www-data | 2020-02-01 15:03:29 +0100 (sam. 01 févr. 2020) | 1 ligne
00004 
00005 Creating initial repository structure
```

8.3 Référence du fichier communication.cpp

Définition de la classe Communication (Module Ecran-DARTS)

```
#include "communication.h"
#include <QDebug>
```

8.3.1 Description détaillée

Définition de la classe Communication (Module Ecran-DARTS)

Auteur

Bounoir Fabien

Version

0.3

Définition dans le fichier communication.cpp.

8.4 communication.cpp

```
00001 #include "communication.h"
00002 #include <QDebug>
00003
00022 Communication::Communication(Darts *darts,
      QObject *parent) : QObject(parent), darts(darts), serveur(nullptr), socket(nullptr),
      localDeviceName("Ecran-Darts"), trame("")
00023 {
00024     qDebug() << Q_FUNC_INFO;
00025
00026     parametrerBluetooth();
00027
00028     miseAJourEtatPartieAttente();
00029 }
00030
00036 Communication::~Communication()
00037 {
00038     arreter();
00039     localDevice.setHostMode(QBluetoothLocalDevice::HostPoweredOff);
00040     qDebug() << Q_FUNC_INFO;
00041 }
00042
00049 int Communication::getEtatPartie()
00050 {
00051     return etatPartie;
00052 }
00053
00059 void Communication::parametrerBluetooth()
00060 {
00061     if (!localDevice.isValid())
00062     {
00063         qDebug() << Q_FUNC_INFO << "Communication Bluetooth locale valide : " <<
      localDevice.isValid();
00064         emit erreurBluetooth("Communication Bluetooth locale valide : " + QString::number(
      localDevice.isValid()));
00065
00066         return;
00067     }
00068     else
00069     {
00070         // active le bluetooth
00071         localDevice.powerOn();
00072
00073         // lire le nom de l'appareil local
00074         localDeviceName = localDevice.name();
00075
00076
00077         //localDevice.setHostMode(QBluetoothLocalDevice::HostConnectable);
00078
00079         //ou
```

```
00080
00081         //les appareil qui ne sont pas jumelé peuvent decouvrir ecran-DARTS
00082         localDevice.setHostMode(QBluetoothLocalDevice::HostDiscoverable);
00083
00084         QList<QBluetoothAddress> remotes;
00085         remotes = localDevice.connectedDevices();
00086
00087         connect(&localDevice, SIGNAL(deviceConnected(QBluetoothAddress)), this,
      SLOT(deviceConnected(QBluetoothAddress)));
00088         connect(&localDevice, SIGNAL(deviceDisconnected(QBluetoothAddress)),
      this, SLOT(deviceDisconnected(QBluetoothAddress)));
00089         connect(&localDevice, SIGNAL(error(QBluetoothLocalDevice::Error)), this, SLOT(
      error(QBluetoothLocalDevice::Error)));
00090
00091         connect(darts, SIGNAL(etatPartieFini()), this , SLOT(
      miseAJourEtatPartieFin()));
00092         connect(darts, SIGNAL(etatPartieTournois()), this , SLOT(
      miseAJourEtatPartieTournois()));
00093         connect(darts, SIGNAL(changerEtatPartie()), this , SLOT(
      miseAJourEtatPartieEnCours()));
00094         connect(darts, SIGNAL(etatPartieAttenteTournois()), this, SLOT(
      miseAJourEtatPartieAttenteTournois()));
00095     }
00096 }
00097
00103 void Communication::demarrer()
00104 {
00105     if (!localDevice.isValid())
00106         return;
00107
00108     if (!serveur)   //Démarre le serveur s'il n'est pas déjà démarré
00109     {
00110         serveur = new QBluetoothServer(QBluetoothServiceInfo::RfcommProtocol, this);
00111         connect(serveur, SIGNAL(newConnection()), this, SLOT(
      nouveauClient()));
00112
00113         QBluetoothUuid uuid = QBluetoothUuid(serviceUuid);
00114         serviceInfo = serveur->listen(uuid, serviceNom);
00115     }
00116 }
00117
00123 void Communication::arreter()
00124 {
00125     if (!localDevice.isValid())
00126         return;
00127
00128     if (!serveur)
00129         return;
00130
00131     serviceInfo.unregisterService();
00132
00133     if (socket)
00134     {
00135         if (socket->isOpen())
00136         {
00137            socket->close();
00138         }
00139         delete socket;
00140         socket = nullptr;
00141     }
00142
00143     delete serveur;
00144     serveur = nullptr;
00145 }
00146
00152 void Communication::nouveauClient()
00153 {
00154     // on récupère la socket
00155     socket = serveur->nextPendingConnection();
00156     if (!socket)
00157         return;
00158
00159     connect(socket, SIGNAL(disconnected()), this, SLOT(socketDisconnected()));
00160     connect(socket, SIGNAL(readyRead()), this, SLOT(socketReadyRead()));
00161 }
00162
00168 void Communication::socketReadyRead()
00169 {
00170     QByteArray donnees;
00171
00172     while (socket->bytesAvailable())
00173     {
00174         donnees += socket->readAll();
00175         usleep(150000); // cf. timeout
00176     }
00177
00178     trame = QString(donnees);
00179
00180     qDebug() << Q_FUNC_INFO << QString::fromUtf8("Trame reçues : ") << QString(donnees);
00181
00182     decomposerTrame();
```

```
00183 }
00184
00190 void Communication::decomposerTrame()
00191 {
00192     if(estValide()) //test si la trame est valide
00193     {
00194         trame.remove(DELIMITEUR_FIN);
00195         if(trame.contains("START") && (etatPartie == EtatPartie::Attente ||
      etatPartie == EtatPartie::Fin))
00196         {
00197             emit resetPartie();
00198             darts->reinitialiserPartie();
00199
00200             QString modeJeu;
00201             QStringList joueurs;
00202
00203             extraireParametresTrameStart(joueurs, modeJeu);
00204         }
00205         else if(trame.contains("GAME") && etatPartie == EtatPartie::EnCours)
00206         {
00207             darts->receptionnerImpact(trame.section(";",2,2).toInt(),
      trame.section(";",3,3).toInt());
00208         }
00209         else if(trame.contains("GAME") && etatPartie == EtatPartie::Tournois)
00210         {
00211             darts->receptionnerImpactTournois(
      trame.section(";",2,2).toInt(), trame.section(";",3,3).toInt());
00212         }
00213         else if(trame.contains("REGLE")&& etatPartie != EtatPartie::Regle)
00214         {
00215             extraireParametresTrameRegle();
00216         }
00217         else if(trame.contains("PAUSE") && (etatPartie == EtatPartie::EnCours ||
      etatPartie == EtatPartie::Tournois))
00218         {
00219             etatPrecedent = etatPartie;
00220             emit pause();
00221             miseAJourEtatPartiePause();
00222         }
00223         else if(trame.contains("PLAY") && etatPartie == EtatPartie::Pause)
00224         {
00225             emit play();
00226             relancerPartie();
00227         }
00228         else if(trame.contains("SON"))
00229         {
00230             emit jouerSon(trame.section(";",2,2));
00231         }
00232         else if(trame.contains("TOURNOIS"))
00233         {
00234             decomposerTrameTournois();
00235         }
00236         else if(trame.contains("RESET")) // quelque soit l'état de la partie    /** $DART;RESET */
00237         {
00238             reamorcerPartie();
00239         }
00240         else if(trame.contains("STOP") && (etatPartie == EtatPartie::EnCours ||
      etatPartie == EtatPartie::Pause))
00241         {
00242             darts->arreterPartie();
00243         }
00244         else if(trame.contains("STOP") && (etatPartie == EtatPartie::Regle))   //permet
       l'arret des regles en cours de lecture
00245         {
00246             emit stopperRegle();
00247         }
00248         else
00249         {
00250             qDebug() << Q_FUNC_INFO << "Trame non Traité: " << trame;
00251         }
00252     }
00253 }
00254
00261 bool Communication::estValide()
00262 {
00263     if(trame.startsWith(TYPE_TRAME) && trame.endsWith(
      DELIMITEUR_FIN) && trame.contains(";"))
00264     {
00265         return true;
00266     }
00267     else
00268     {
00269         qDebug() << Q_FUNC_INFO << "Trame non Valide: " << trame;
00270         return false;
00271     }
00272 }
00273
00281 void Communication::extraireParametresTrameStart(QStringList &
      joueurs, QString &modeJeu)
00282 {
00283     modeJeu = trame.section(";",2,2);
```

```
00284
00285     if(trame.section(";",4,4).toInt() == 0) //test effectuer pour verifier que la trame n'est pas une
      trame de la version du protocol DARTS 0.2
00286         return;
00287
00288     for(int i = 0;i <= trame.section(";",4,4).toInt();i++)  //boucle qui recuperer les noms des
      differents joueurs
00289     {
00290         if(trame.section(";",4+i,4+i) == "")    //test si le joueur a un nom
00291         {
00292             joueurs.push_back("Joueur[" + QString::number(i) + "]");
00293         }
00294         else
00295         {
00296             joueurs.push_back(trame.section(";",4+i,4+i));
00297         }
00298     }
00299
00300     darts->initialiserPartie(joueurs, modeJeu);
00301
00302     if(trame.section(";",3,3) == "1")
00303     {
00304         if(etatPartie != EtatPartie::Pause)
00305             emit pause();
00306
00307         emit afficherRegle(darts->testerModeDeJeu());
00308     }
00309
00310 }
00311
00317 void Communication::decomposerTrameTournois()
00318 {
00319     QString modeJeu;
00320     QString nomTournois;
00321     QStringList joueurs;
00322
00323     if(trame.contains("CONFIG") && (etatPartie == EtatPartie::Attente ||
      etatPartie == EtatPartie::Fin))
00324     {
00325         modeJeu = trame.section(";",3,3);
00326
00327         if(trame.section(";",5,5).toInt()%2 !=0)
00328         {
00329             qDebug() << "Nombre de joueur insuffissant";
00330             return;
00331         }
00332
00333         darts->reinitialiserPartie();
00334
00335         for(int i = 0;i <= trame.section(";",5,5).toInt();i++)  //boucle qui recuperer les noms des
      differents joueurs
00336         {
00337             if(trame.section(";",5+i,5+i) == "")    //test si le joueur a un nom
00338             {
00339                 joueurs.push_back("Joueur[" + QString::number(i) + "]");
00340             }
00341             else
00342             {
00343                 joueurs.push_back(trame.section(";",5+i,5+i));
00344             }
00345         }
00346         nomTournois = trame.section(";",4,4);
00347         darts->configurationTournois(joueurs, modeJeu, nomTournois);
00348
00349     }
00350     else if(trame.contains("PLAY") && (etatPartie == EtatPartie::AttenteTournois))
00351     {
00352         darts->demarrerTournois();
00353     }
00354     else
00355     {
00356         qDebug() << Q_FUNC_INFO << "Trame non Traité: " << trame;
00357     }
00358 }
00359
00365 void Communication::extraireParametresTrameRegle()
00366 {
00367     QString regle ="";
00368
00369     if(etatPartie != EtatPartie::Pause)
00370         emit pause();
00371
00372     if(trame.section(";",2,2).contains("DOUBLE_OUT"))
00373     {
00374         emit afficherRegle("DOUBLE_OUT");
00375     }
00376     else if(trame.section(";",2,2) == "" && (etatPartie == EtatPartie::EnCours ||
      etatPartie == EtatPartie::Pause))
00377     {
00378         emit afficherRegle(darts->testerModeDeJeu());
00379     }
```

```
00380     else
00381     {
00382         emit afficherRegle("SANS_DOUBLE_OUT");
00383     }
00384 }
00385
00391 void Communication::reamorcerPartie()
00392 {
00393     emit resetPartie();
00394     darts->reinitialiserPartie();
00395     miseAJourEtatPartieAttente();
00396 }
00397
00403 void Communication::socketDisconnected()
00404 {
00405     QString message = QString::fromUtf8("Périphérique déconnecté ");
00406     qDebug() << Q_FUNC_INFO << message;
00407 }
00408
00415 void Communication::deviceConnected(const QBluetoothAddress &adresse)
00416 {
00417     qDebug() << Q_FUNC_INFO << adresse << localDevice.pairingStatus(adresse);
00418     QString message = QString::fromUtf8("Demande connexion du client ") + adresse.toString();
00419     emit appareilConnecter();
00420     if (localDevice.pairingStatus(adresse) == QBluetoothLocalDevice::Paired ||
      localDevice.pairingStatus(adresse) == QBluetoothLocalDevice::AuthorizedPaired)
00421         message += " [" + QString::fromUtf8("appairé") + "]";
00422     else
00423         message += " [" + QString::fromUtf8("non appairé") + "]" ;
00424     qDebug() << message << endl;
00425
00426     if(etatPartie == EtatPartie::Pause) // si l'appareil est reconnecte, la partie reprend
00427     {
00428         emit play();
00429         relancerPartie();
00430     }
00431 }
00432
00439 void Communication::deviceDisconnected(const QBluetoothAddress &adresse)
00440 {
00441     qDebug() << Q_FUNC_INFO << adresse;
00442     emit afficherAttenteConnexion();
00443
00444     if(etatPartie == EtatPartie::EnCours || etatPartie == EtatPartie::Tournois) // si
      l'appareil se deconnecte pendant la partie, il la met donc en pause
00445     {
00446         etatPrecedent = etatPartie;
00447         emit pause();
00448         miseAJourEtatPartiePause();
00449     }
00450 }
00451
00458 void Communication::error(QBluetoothLocalDevice::Error erreur)
00459 {
00460     qDebug() << Q_FUNC_INFO << erreur;
00461 }
00462
00468 void Communication::miseAJourEtatPartieAttente()
00469 {
00470     qDebug() << Q_FUNC_INFO << "EtatPartie::Attente";
00471     etatPartie = EtatPartie::Attente;
00472 }
00473
00479 void Communication::miseAJourEtatPartiePause()
00480 {
00481     qDebug() << Q_FUNC_INFO << "EtatPartie::Pause";
00482     etatPartie = EtatPartie::Pause;
00483 }
00484
00490 void Communication::miseAJourEtatPartieFin()
00491 {
00492     qDebug() << Q_FUNC_INFO << "EtatPartie::Fin";
00493     etatPartie = EtatPartie::Fin;
00494 }
00495
00501 void Communication::miseAJourEtatPartieEnCours()
00502 {
00503     qDebug() << Q_FUNC_INFO << "EtatPartie::EnCours";
00504     etatPartie = EtatPartie::EnCours;
00505 }
00506
00512 void Communication::miseAJourEtatPartieRegle()
00513 {
00514     qDebug() << Q_FUNC_INFO << "EtatPartie::Regle";
00515     etatPartie = EtatPartie::Regle;
00516 }
00517
00523 void Communication::miseAJourEtatPartieTournois()
00524 {
00525     qDebug() << Q_FUNC_INFO << "EtatPartie::Tournois";
00526     etatPartie = EtatPartie::Tournois;
```

```
00527 }
00528
00529
00535 void Communication::miseAJourEtatPartieAttenteTournois()
00536 {
00537     qDebug() << Q_FUNC_INFO << "EtatPartie::AttenteTournois";
00538     etatPartie = EtatPartie::AttenteTournois;
00539 }
00540
00546 void Communication::relancerPartie()
00547 {
00548     if(etatPrecedent == EtatPartie::EnCours)
00549     {
00550         miseAJourEtatPartieEnCours();
00551     }
00552     else
00553     {
00554         miseAJourEtatPartieTournois();
00555     }
00556 }
```

8.5 Référence du fichier communication.h

Déclaration de la classe Communication (Module Ecran-DARTS)

```
#include "darts.h"
#include <QObject>
#include <QString>
#include <QBluetoothLocalDevice>
#include <QBluetoothServer>
#include <unistd.h>
```

Classes

— class Communication
Déclaration de la classe Communication via la liaison Bluetooth (Module Ecran-DARTS)

Macros

— #define DELIMITEUR_FIN "\r\n"
Définit le délimiteur de fin de trame du protocole DARTS.
— #define TYPE_TRAME "$DART"
Définit le type de trame du protocole DARTS.

Fonctions

— static const QString serviceNom (QStringLiteral("Ecran-Darts"))
— static const QString serviceUuid (QStringLiteral("0000110a-0000-1000-8000-00805f9b34fb"))

8.5.1 Description détaillée

Déclaration de la classe Communication (Module Ecran-DARTS)

Version

0.3

Auteur

Bounoir Fabien

Définition dans le fichier communication.h.

8.5.2 Documentation des macros

8.5.2.1 DELIMITEUR_FIN

```
#define DELIMITEUR_FIN "\r\n"
```

Définit le délimiteur de fin de trame du protocole DARTS.

Définition à la ligne 30 du fichier communication.h.

Référencé par Communication : :decomposerTrame(), et Communication : :estValide().

8.5.2.2 TYPE_TRAME

```
#define TYPE_TRAME "$DART"
```

Définit le type de trame du protocole DARTS.

Définition à la ligne 24 du fichier communication.h.

Référencé par Communication : :estValide().

8.5.3 Documentation des fonctions

8.5.3.1 serviceNom()

```
static const QString serviceNom (
            QStringLiteral("Ecran-Darts")  )  [static]
```

Référencé par Communication : :demarrer().

8.5.3.2 serviceUuid()

```
static const QString serviceUuid (
            QStringLiteral("0000110a-0000-1000-8000-00805f9b34fb")  )  [static]
```

Référencé par Communication : :demarrer().

8.6 communication.h

```
00001 #ifndef COMMUNICATION_H
00002 #define COMMUNICATION_H
00003
00013 #include "darts.h"
00014 #include <QObject>
00015 #include <QString>
00016 #include <QBluetoothLocalDevice>
00017 #include <QBluetoothServer>
00018 #include <unistd.h>
00019
00024 #define TYPE_TRAME "$DART"
00025
00030 #define DELIMITEUR_FIN "\r\n"
00031
00032 static const QString serviceUuid(QStringLiteral("0000110a-0000-1000-8000-00805f9b34fb"));
00033 static const QString serviceNom(QStringLiteral("Ecran-Darts"));
00034
00041 class Communication : public QObject
00042 {
00043     Q_OBJECT
00044 public:
00045     explicit Communication(Darts *darts, QObject *parent = nullptr);
00046     ~Communication();
00047
00048     void parametrerBluetooth();
00049     void demarrer();
00050     void arreter();
00051     int getEtatPartie();
00052
00053     void miseAJourEtatPartieRegle();
00054     void miseAJourEtatPartieAttente();
00055     void miseAJourEtatPartiePause();
00056
00061     enum EtatPartie
00062     {
00063         Attente = 0,
00064         EnCours = 1,
00065         Fin = 2,
00066         Pause = 3,
00067         Regle = 4,      //État attente pendant la lecture des règles, seule une trame reset peut être
      reçu pour arrêter la lecture.
00068         AtteneTournois = 5,
00069         Tournois
00070     };
00071
00072 signals:
00073     void appareilConnecter();
00074     void afficherAtteneConnexion();
00075     void resetPartie();
00076     void pause();
00077     void play();
00078     void erreurBluetooth(QString erreur);
00079     void afficherRegle(QString regle);
00080     void stopperRegle();
00081     void jouerSon(QString);
00082
00083 public slots:
00084     void miseAJourEtatPartieFin();
00085     void miseAJourEtatPartieEnCours();
00086     void miseAJourEtatPartieTournois();
00087     void miseAJourEtatPartieAtteneTournois();
00088
00089 private slots:
00090     void deviceConnected(const QBluetoothAddress &adresse);
00091     void deviceDisconnected(const QBluetoothAddress &adresse);
00092     void error(QBluetoothLocalDevice::Error erreur);
00093     void nouveauClient();
00094     void socketReadyRead();
00095     void socketDisconnected();
00096
00097
00098 private:
00099     Darts *darts;
00100     QBluetoothServer *serveur;
00101     QBluetoothSocket *socket;
00102     QBluetoothLocalDevice localDevice;
00103     QBluetoothServiceInfo serviceInfo;
00104     QString localDeviceName;
00105     QString trame;
00106     EtatPartie etatPartie;
00107
00108     void decomposerTrame();
00109     void extraireParametresTrameStart(QStringList &joueurs, QString &modeJeu);

00110     void extraireParametresTrameRegle();

00111     void reamorcerPartie();
00112     void decomposerTrameTournois();
```

```
00113     bool estValide();
00114     int etatPrecedent;
00115     void relancerPartie();
00116 };
00117
00118 #endif // COMMUNICATION_H
```

8.7 Référence du fichier darts.cpp

Définition de la classe Darts (Module Ecran-DARTS)

```
#include "darts.h"
#include <QDebug>
```

8.7.1 Description détaillée

Définition de la classe Darts (Module Ecran-DARTS)

Auteur

Bounoir Fabien

Version

0.3

Définition dans le fichier darts.cpp.

8.8 darts.cpp

```
00001  #include "darts.h"
00002
00003 #include <QDebug>
00004
00022 Darts::Darts(QObject *parent) : QObject(parent), joueur(nullptr), nbJoueur(0),
      joueurActif(0), manche(1), pointLancer(0), voleeMax(0), nbVolees(0), ModeDeJeu(""),pointVoleeEnCours(0)
00023 {
00024     qDebug() << Q_FUNC_INFO;
00025     solution = new Solution(this);
00026 }
00027
00033 Darts::~Darts()
00034 {
00035     qDebug() << Q_FUNC_INFO;
00036 }
00037
00044 int Darts::getManche() const
00045 {
00046     return manche;
00047 }
00048
00055 int Darts::getPointVolees() const
00056 {
00057     return pointVoleeEnCours;
00058 }
00059
00066 QList<Joueur> Darts::getListJoueur() const
00067 {
00068     return joueurs;
00069 }
00070
00077 int Darts::getVoleeMax() const
00078 {
00079     return voleeMax;
00080 }
00081
00088 int Darts::getJoueurActif() const
00089 {
00090     return joueurActif;
```

```
00091 }
00092
00099 int Darts::getNbVolees() const
00100 {
00101     return nbVolees;
00102 }
00103
00110 int Darts::getPremierJoueur() const
00111 {
00112     return premierJoueur;
00113 }
00114
00121 int Darts::getDernierJoueur() const
00122 {
00123     return dernierJoueur;
00124 }
00125
00132 QString Darts::getModeDeJeu() const
00133 {
00134     return ModeDeJeu;
00135 }
00136
00143 Solution *Darts::getSolution() const
00144 {
00145     return solution;
00146 }
00147
00154 void Darts::setVoleeMax(int voleeMax)
00155 {
00156     this->voleeMax = voleeMax;
00157 }
00158
00165 void Darts::setManche(int manche)
00166 {
00167     this->manche = manche;
00168 }
00169
00177 void Darts::initialiserPartie(QStringList joueurList, QString modeJeu)
00178 {
00179     nbJoueur = joueurList.size() - 1;
00180     ModeDeJeu = modeJeu;
00181     qDebug() << Q_FUNC_INFO << "nbJoueur " << nbJoueur << " | Mode De Jeu " <<
    ModeDeJeu;
00182
00183     if(ModeDeJeu.toInt() >= 101 && ModeDeJeu.toInt() <= 1001)
00184     {
00185         for(int i = 1; i < joueurList.size() ; i++)
00186         {
00187             Joueur player(joueurList.at(i), ModeDeJeu.toInt(),
    NB_FLECHETTE);
00188             joueurs.push_back(player);
00189         }
00190         emit afficherNouvellePartie();
00191         emit changerEtatPartie();
00192     }
00193     else if(ModeDeJeu.contains("_DOUBLE_OUT") && (ModeDeJeu.section("_",0,0).toInt() >= 101 && ModeDeJeu.
    section("_",0,0).toInt() <= 1001))
00194     {
00195         for(int i = 1; i < joueurList.size() ; i++)
00196         {
00197             Joueur player(joueurList.at(i), ModeDeJeu.section("_",0,0).toInt(),
    NB_FLECHETTE);
00198             joueurs.push_back(player);
00199         }
00200         emit afficherNouvellePartie();
00201         emit changerEtatPartie();
00202     }
00203     else
00204     {
00205         qDebug() << Q_FUNC_INFO << "Erreur Mode De Jeu : " << ModeDeJeu;
00206         reinitialiserPartie();
00207         return;
00208     }
00209     solution->trouverSolution(joueurs[joueurActif].getScore(),
    joueurs[joueurActif].getFlechette());
00210 }
00211
00217 void Darts::reinitialiserPartie()
00218 {
00219     joueurs.clear();
00220     joueur.clear();
00221     joueursTournoisEliminer.clear();
00222     premierJoueur = 0;
00223     dernierJoueur = 1;
00224
00225     ModeDeJeu = "";
00226     nbJoueur = 0;
00227     joueurActif = 0;
00228     manche = 1;
00229     pointLancer = 0;
00230     nbVolees = 0;
```

```
00231     voleeMax = 0;
00232     pointVoleeEnCours = 0;
00233 }
00234
00242 void Darts::testerPoint(int typePoint, int point)
00243 {
00244     if(typePoint == DOUBLE_POINT && point == BULL)
00245     {
00246         jouerSon("D25.wav");
00247     }
00248     else if(typePoint == ZERO_POINT)
00249     {
00250         jouerSon("out.wav");
00251     }
00252 }
00253
00261 void Darts::receptionnerImpact(int typePoint, int point)
00262 {
00263     calculerPoints(point, typePoint);
00264
00265     testerPoint(typePoint, point);
00266
00267     pointVoleeEnCours += pointLancer;
00268
00269     if(joueurs[joueurActif].getFlechette() == NB_FLECHETTE)
00270         emit actualiserCible();
00271
00272     qDebug() << Q_FUNC_INFO << joueurs[joueurActif].getNom() << " SCORE : " <<
      joueurs[joueurActif].getScore() - pointLancer << endl;
00273
00274     emit nouvelImpact(typePoint, point, pointLancer);
00275     joueurs[joueurActif].setScore(joueurs[joueurActif].getScore() -
      pointLancer);
00276     testerImpact(typePoint);
00277     emit miseAJourPoint();
00278 }
00279
00286 void Darts::testerImpact(int typePoint)
00287 {
00288     if(joueurs[joueurActif].getScore()  == 0 && (typePoint ==
      DOUBLE_POINT) && (ModeDeJeu.contains("_DOUBLE_OUT"))) //fin avec double
00289     {
00290         gererVoleeMax();
00291         nbVolees++;
00292         emit finPartie("  Winner " + joueurs[joueurActif].getNom() + "  ",
      getVoleeMax(), false);
00293         emit etatPartieFini();
00294     }
00295     else if(joueurs[joueurActif].getScore()  == 0 && !
      ModeDeJeu.contains("_DOUBLE_OUT"))    //fin sans double
00296     {
00297         gererVoleeMax();
00298         nbVolees++;
00299         emit finPartie("  Winner " + joueurs[joueurActif].getNom() + "  " ,
      getVoleeMax(), false);
00300         emit etatPartieFini();
00301     }
00302     else
00303     {
00304         enleverPointImpact();
00305         gererManche();
00306     }
00307 }
00308
00314 void Darts::enleverPointImpact()
00315 {
00316     if(joueurs[joueurActif].getScore() <= 0) //score Volée inferieure au score = Volée
       annulée
00317     {
00318         joueurs[joueurActif].setScore(joueurs[
      joueurActif].getScoreManchePrecedente());
00319         emit voleeAnnulee();
00320
00321         joueurs[joueurActif].setNbFlechette(0);
00322     }
00323     else if(joueurs[joueurActif].getScore() == 1 && joueurs.size() == 1)    //
       test si le joueur est seul à jouer et s'il est a 1 point == joueur éliminer donc fin de partie
00324     {
00325         gererVoleeMax();
00326         emit jouerSon("perdu.wav");
00327         emit finPartie("  " + joueurs[joueurActif].getNom() + " a perdu  ",
      getVoleeMax(), false);
00328         emit etatPartieFini();
00329     }
00330     else
00331     {
00332         nbVolees++;
00333         joueurs[joueurActif].setNbFlechette(joueurs[
      joueurActif].getFlechette() - 1);
00334         solution->trouverSolution(joueurs[
      joueurActif].getScore(),joueurs[joueurActif].getFlechette());
```

```
00335     }
00336 }
00337
00343 void Darts::gererManche()
00344 {
00345     if(joueurs[joueurActif].getFlechette() == 0) //fin de la volées du joueur
00346     {
00347         joueurs[joueurActif].setNbFlechette(NB_FLECHETTE);
00348
00349         pointVoleeEnCours = 0;
00350
00351         gererVoleeMax();
00352         testerSiJoueurEliminer();
00353         testerNombreJoueurRestand();
00354
00355         joueurs[joueurActif].addHistoriqueVolees((joueurs[
      joueurActif].getScoreManchePrecedente() - joueurs[joueurActif].getScore()));
00356
00357         if((joueurs[joueurActif].getScoreManchePrecedente() -
      joueurs[joueurActif].getScore()) == 180)
00358             emit jouerSon("180.wav");
00359
00360         joueurs[joueurActif].setScoreManchePrecedente(joueurs[
      joueurActif].getScore());
00361
00362         if(joueurActif == nbJoueur - 1)  //equivalent a la fin de manche
00363         {
00364             joueurActif = 0;
00365
00366             controlerJoueurEliminer();
00367
00368             setManche(getManche() + 1);
00369             emit changementJoueurActif();
00370             calculerMoyenneVolees();
00371             emit nouvelleManche();
00372             solution->trouverSolution(joueurs[
      joueurActif].getScore(),joueurs[joueurActif].getFlechette());
00373         }
00374         else
00375         {
00376             joueurActif++;
00377
00378             controlerJoueurEliminer();
00379
00380             emit changementJoueurActif();
00381             solution->trouverSolution(joueurs[
      joueurActif].getScore(),joueurs[joueurActif].getFlechette());
00382         }
00383     }
00384 }
00385
00391 void Darts::controlerJoueurEliminer()
00392 {
00393     while(joueurs[joueurActif].getEliminer()) //tant que le joueur est eliminer on passe
       donc au joueur suivant
00394     {
00395         if(joueurActif == nbJoueur - 1)  //si le joueur est le dernier de la manche à
       jouer
00396         {
00397             joueurActif = 0;
00398             setManche(getManche() + 1);
00399             calculerMoyenneVolees();
00400             emit nouvelleManche();
00401         }
00402         else                            //si le joueur n'est pas le dernier de la manche
00403         {
00404             joueurActif++;
00405         }
00406     }
00407 }
00408
00414 void Darts::testerSiJoueurEliminer()
00415 {
00416     if(joueurs[joueurActif].getScore() == 1)    //joueur eliminé si tomber à 1 point
00417     {
00418         joueurs[joueurActif].setEliminer(true);
00419     }
00420 }
00421
00427 void Darts::calculerMoyenneVolees()
00428 {
00429     for(int i = 0; i <= joueurs.size() - 1; i++)
00430     {
00431         int moyenneVolee = 0;
00432
00433         for(int j = 0; j < joueurs[i].getHistoriqueVolees().size(); j++)
00434         {
00435             moyenneVolee += joueurs[i].getHistoriqueVolees()[j];
00436         }
00437
00438         moyenneVolee = moyenneVolee / joueurs[i].getHistoriqueVolees().size();
```

```
00439         joueurs[i].setMoyenneVolee(moyenneVolee);
00440     }
00441     emit miseAJourMoyenneVolee();
00442 }
00443
00449 void Darts::gererVoleeMax()
00450 {
00451     if((joueurs[joueurActif].getScoreManchePrecedente() -
      joueurs[joueurActif].getScore()) > getVoleeMax())
00452     {
00453         setVoleeMax(joueurs[joueurActif].getScoreManchePrecedente() -
      joueurs[joueurActif].getScore());
00454     }
00455 }
00456
00462 void Darts::arreterPartie()
00463 {
00464     emit finPartie(calculerGagnant(), getVoleeMax(), false);
00465     emit etatPartieFini();
00466 }
00467
00474 QString Darts::calculerGagnant()
00475 {
00476     QString gagnant;
00477     int scoreGagnant = 99999;
00478     for(int i = 0; i <= joueurs.size() - 1; i++)
00479     {
00480         if(scoreGagnant > joueurs[i].getScore() && !joueurs[i].getEliminer()) // test le
       personne aillant le score le plus bas score mais aussi qu'il ne soit pas eliminer
00481         {
00482             scoreGagnant = joueurs[i].getScore();
00483             gagnant = joueurs[i].getNom();
00484         }
00485     }
00486     return "  Winner " + gagnant + "  ";
00487 }
00488
00494 void Darts::testerNombreJoueurRestand()
00495 {
00496     int eliminer = 0;
00497
00498     for(int i = 0; i <= joueurs.size() - 1; i++)
00499     {
00500         if(joueurs[i].getEliminer())
00501         {
00502             eliminer++;
00503         }
00504     }
00505
00506     if(eliminer == joueurs.size() - 1 && joueurs.size() != 1)
00507     {
00508         arreterPartie();
00509     }
00510 }
00511
00519 void Darts::calculerPoints(int point, int typePoint)
00520 {
00521     switch(typePoint)
00522     {
00523         case TRIPLE_POINT:
00524             pointLancer = point * TRIPLE_POINT;
00525         break;
00526         case DOUBLE_POINT:
00527             pointLancer = point * DOUBLE_POINT;
00528         break;
00529         case SIMPLE_POINT:
00530             pointLancer = point;
00531         break;
00532         case  ZERO_POINT:
00533             pointLancer = point * ZERO_POINT;
00534         break;
00535         default:
00536             qDebug() << Q_FUNC_INFO << "type de point non valide:" << typePoint;
00537         break;
00538     }
00539 }
00540
00547 QString Darts::testerModeDeJeu()
00548 {
00549     QString regle ="";
00550
00551     if(getModeDeJeu().contains("DOUBLE_OUT"))
00552     {
00553         regle = "DOUBLE_OUT";
00554     }
00555     else
00556     {
00557         regle = "SANS_DOUBLE_OUT";
00558     }
00559     return regle;
00560 }
```

```
00561
00570 void Darts::configurationTournois(QStringList joueurList, QString modeJeu,
      QString nomTournois)
00571 {
00572     nbJoueur = joueurList.size() - 1;
00573     ModeDeJeu = modeJeu;
00574     NomTournois = nomTournois;
00575     qDebug() << Q_FUNC_INFO << "Nom Tournois " << nomTournois << "nbJoueur " <<
      nbJoueur << " | Mode De Jeu " << ModeDeJeu;
00576
00577     if(ModeDeJeu.toInt() >= 101 && ModeDeJeu.toInt() <= 1001)
00578     {
00579         for(int i = 1; i < joueurList.size() ; i++)
00580         {
00581             Joueur player(joueurList.at(i), ModeDeJeu.toInt(),
      NB_FLECHETTE);
00582             joueurs.push_back(player);
00583         }
00584     }
00585     else if(ModeDeJeu.contains("_DOUBLE_OUT") && (ModeDeJeu.section("_",0,0).toInt() >= 101 && ModeDeJeu.
      section("_",0,0).toInt() <= 1001))
00586     {
00587         for(int i = 1; i < joueurList.size() ; i++)
00588         {
00589             Joueur player(joueurList.at(i), ModeDeJeu.section("_",0,0).toInt(),
      NB_FLECHETTE);
00590             joueurs.push_back(player);
00591         }
00592     }
00593     else
00594     {
00595         qDebug() << Q_FUNC_INFO << "Erreur Mode De Jeu : " << ModeDeJeu;
00596         reinitialiserPartie();
00597         return;
00598     }
00599     solution->trouverSolution(joueurs[joueurActif].getScore(),
      joueurs[joueurActif].getFlechette());
00600
00601     premierJoueur = 0;
00602     dernierJoueur = 1;
00603
00604     emit afficherTournois(modeJeu, nomTournois);
00605     emit etatPartieAttenteTournois();
00606     emit afficherInfoTournois();
00607 }
00608
00614 void Darts::demarrerTournois()
00615 {
00616     emit actualiserCible();
00617     emit changementJoueurActifTournoi();
00618     emit debuterTournois();
00619     emit etatPartieTournois();
00620 }
00621
00629 void Darts::receptionnerImpactTournois(int typePoint, int point)
00630 {
00631     calculerPoints(point, typePoint);
00632
00633     testerPoint(typePoint, point);
00634
00635     pointVoleeEnCours += pointLancer;
00636
00637     if(joueurs[joueurActif].getFlechette() == NB_FLECHETTE)
00638         emit actualiserCible();
00639
00640     emit nouvelImpact(typePoint, point, pointLancer);
00641
00642     joueurs[joueurActif].setScore(joueurs[joueurActif].getScore() -
      pointLancer);
00643
00644     testerImpactTournois(typePoint);
00645     emit miseAJourPointTournois();
00646 }
00647
00654 void Darts::testerImpactTournois(int typePoint)
00655 {
00656     if(joueurs[joueurActif].getScore()  == 0 && (typePoint ==
      DOUBLE_POINT) && (ModeDeJeu.contains("_DOUBLE_OUT"))) //fin avec double
00657     {
00658         gererVoleeMax();
00659         nbVolees++;
00660         emit afficherInfoTournois();
00661         emit finPartie("  " + joueurs[joueurActif].getNom() +" Winner de la
       manche" +"  " , getVoleeMax(), true);
00662         gererFinPartieTournois();
00663         emit etatPartieAttenteTournois();
00664     }
00665     else if(joueurs[joueurActif].getScore()  == 0 && !
      ModeDeJeu.contains("_DOUBLE_OUT"))    //fin sans double
00666     {
00667         gererVoleeMax();
```

```
00668         nbVolees++;
00669         emit afficherInfoTournois();
00670         emit finPartie("  " + joueurs[joueurActif].getNom() +" Winner de la
       manche" +"  " , getVoleeMax(), true);
00671         gererFinPartieTournois();
00672         emit etatPartieAttenteTournois();
00673     }
00674     else
00675     {
00676         enleverPointImpact();
00677         gererMancheTournois();
00678     }
00679 }
00680
00681
00687 void Darts::gererMancheTournois()
00688 {
00689     if(joueurs[joueurActif].getFlechette() == 0) //fin de la volées du joueur
00690     {
00691         joueurs[joueurActif].setNbFlechette(NB_FLECHETTE);
00692
00693         pointVoleeEnCours = 0;
00694
00695         gererVoleeMax();
00696         //testerSiJoueurEliminer(); /** @todo changer pour faire gagner l'autre joueur*/
00697
00698         joueurs[joueurActif].addHistoriqueVolees((joueurs[
      joueurActif].getScoreManchePrecedente() - joueurs[joueurActif].getScore()));
00699
00700         if((joueurs[joueurActif].getScoreManchePrecedente() -
      joueurs[joueurActif].getScore()) == 180)
00701             emit jouerSon("180.wav");
00702
00703         joueurs[joueurActif].setScoreManchePrecedente(joueurs[
      joueurActif].getScore());
00704
00705         if(joueurActif == dernierJoueur)  //equivalent a la fin de manche
00706         {
00707             joueurActif = premierJoueur;
00708
00709             setManche(getManche() + 1);
00710             emit changementJoueurActifTournoi();
00711             calculerMoyenneVoleesTournois();
00712             emit nouvelleManche();
00713             solution->trouverSolution(joueurs[
      joueurActif].getScore(),joueurs[joueurActif].getFlechette());
00714         }
00715         else
00716         {
00717             joueurActif++;
00718
00719             emit changementJoueurActifTournoi();
00720             solution->trouverSolution(joueurs[
      joueurActif].getScore(),joueurs[joueurActif].getFlechette());
00721         }
00722     }
00723 }
00724
00730 void Darts::calculerMoyenneVoleesTournois()
00731 {
00732     for(int i = premierJoueur; i <= dernierJoueur; i++)
00733     {
00734         int moyenneVolee = 0;
00735
00736         for(int j = 0; j < joueurs[i].getHistoriqueVolees().size(); j++)
00737         {
00738             moyenneVolee += joueurs[i].getHistoriqueVolees()[j];
00739         }
00740
00741         moyenneVolee = moyenneVolee / joueurs[i].getHistoriqueVolees().size();
00742         joueurs[i].setMoyenneVolee(moyenneVolee);
00743     }
00744     emit miseAJourMoyenneVoleeTournois();
00745 }
00746
00752 void Darts::gererFinPartieTournois()
00753 {
00754     if(joueurActif == premierJoueur)
00755     {
00756         joueurs[premierJoueur].setEliminer(false);
00757         joueurs[dernierJoueur].setEliminer(true);
00758     }
00759
00760     if(joueurActif == dernierJoueur)
00761     {
00762         joueurs[dernierJoueur].setEliminer(false);
00763         joueurs[premierJoueur].setEliminer(true);
00764     }
00765
00766     if(premierJoueur + 2 < (joueurs.size() - 1)  &&
      dernierJoueur + 2 <= (joueurs.size() - 1))
```

```
00767     {
00768         premierJoueur += 2;
00769         dernierJoueur += 2;
00770         nbVolees = 0;
00771 
00772         qDebug() << "premierJoueur " << premierJoueur << endl;
00773         qDebug() << "dernierJoueur " << dernierJoueur << endl;
00774     }
00775     else if(estDernier())
00776     {
00777         qDebug() << "Gagnant tournois " << joueurs[joueurActif].getNom() << endl;
00778         initialiserFinTournois();
00779         emit etatPartieFini();
00780     }
00781     else
00782     {
00783         for(int i = joueurs.size() - 1; i >= 0; i--)
00784         {
00785             if(joueurs[i].getEliminer())
00786             {
00787                 joueursTournoisEliminer.push_back(joueurs[i]);
00788                 joueurs.removeAt(i);
00789             }
00790         }
00791         premierJoueur = 0;
00792         dernierJoueur = 1;
00793         nbVolees = 0;
00794 
00795         for(int i = 0; i < joueurs.size(); i++)
00796         {
00797 
00798             joueurs[i].setScore(ModeDeJeu.section("_",0,0).toInt());
00799             joueurs[i].setMoyenneVolee(0);
00800             joueurs[i].setScoreManchePrecedente(ModeDeJeu.section("_",0,0).toInt());
00801             joueurs[i].setNbFlechette(NB_FLECHETTE);
00802         }
00803     }
00804 
00805     joueurActif = premierJoueur;
00806     setManche(1);
00807 
00808     emit afficherTournois(ModeDeJeu, NomTournois);
00809 
00810 }
00811 
00818 bool Darts::estDernier()
00819 {
00820     int estdernier = 0;
00821     for(int i = 0; i < joueurs.size(); i++)
00822     {
00823         qDebug() << joueurs[i].getNom() << " : " << joueurs[i].getEliminer() << endl;
00824         if(joueurs[i].getEliminer() == false)
00825             estdernier++;
00826     }
00827 
00828     if(estdernier == 1)
00829         return true;
00830 
00831     return false;
00832 }
00833 
00839 void Darts::initialiserFinTournois()
00840 {
00841     QList<Joueur> joueurTournois;
00842 
00843     for(int i = 0 ; i < joueurs.size(); i++)
00844     {
00845         joueurTournois.push_back(joueurs[i]);
00846     }
00847 
00848     for(int i = 0 ; i < joueursTournoisEliminer.size(); i++)
00849     {
00850         joueurTournois.push_back(joueursTournoisEliminer[i]);
00851     }
00852 
00853     emit finTournois(joueurs[joueurActif].getNom(),
      NomTournois,joueurTournois);
00854 }
```

8.9 Référence du fichier darts.h

Déclaration de la classe Darts (Module Ecran-DARTS)

```
#include "joueur.h"
#include "solution.h"
```

```
#include <QObject>
#include <QVector>
```

Classes

— class Darts
Déclaration de la classe Darts (Module Ecran-DARTS)

Macros

— #define BULL 25
— #define DOUBLE_POINT 2
— #define NB_FLECHETTE 3
— #define SIMPLE_POINT 1
— #define TRIPLE_POINT 3
— #define ZERO_POINT 0

8.9.1 Description détaillée

Déclaration de la classe Darts (Module Ecran-DARTS)

Version

0.3

Auteur

Bounoir Fabien

Définition dans le fichier darts.h.

8.9.2 Documentation des macros

8.9.2.1 BULL

```
#define BULL 25
```

Définition à la ligne 22 du fichier darts.h.

Référencé par Solution : :rechercherDouble(), Solution : :rechercherSimple(), Darts : :testerPoint(), et Solution : :trouverSolution().

8.9.2.2 DOUBLE_POINT

```
#define DOUBLE_POINT 2
```

Définition à la ligne 19 du fichier darts.h.

Référencé par Darts : :calculerPoints(), Ihm : :mettreAJourMessageStatut(), Darts : :testerImpact(), Darts : :testerImpactTournois(), et Darts : :testerPoint().

8.9.2.3 NB_FLECHETTE

```
#define NB_FLECHETTE 3
```

Définition à la ligne 23 du fichier darts.h.

Référencé par Darts : :configurationTournois(), Darts : :gererFinPartieTournois(), Darts : :gererManche(), Darts : :gererManche↩Tournois(), Darts : :initialiserPartie(), Darts : :receptionnerImpact(), et Darts : :receptionnerImpactTournois().

8.9.2.4 SIMPLE_POINT

```
#define SIMPLE_POINT 1
```

Définition à la ligne 20 du fichier darts.h.

Référencé par Darts : :calculerPoints().

8.9.2.5 TRIPLE_POINT

```
#define TRIPLE_POINT 3
```

Définition à la ligne 18 du fichier darts.h.

Référencé par Darts : :calculerPoints(), et Ihm : :mettreAJourMessageStatut().

8.9.2.6 ZERO_POINT

```
#define ZERO_POINT 0
```

Définition à la ligne 21 du fichier darts.h.

Référencé par Darts : :calculerPoints(), Ihm : :mettreAJourMessageStatut(), et Darts : :testerPoint().

8.10 darts.h

```
00001 #ifndef DARTS_H
00002 #define DARTS_H
00003
00013 #include "joueur.h"
00014 #include "solution.h"
00015 #include <QObject>
00016 #include <QVector>
00017
00018 #define TRIPLE_POINT        3
00019 #define DOUBLE_POINT        2
00020 #define SIMPLE_POINT        1
00021 #define ZERO_POINT          0
00022 #define BULL                25
00023 #define NB_FLECHETTE        3
00024
00025 class Joueur;
00026 class Solution;
00027
00033 class Darts : public QObject
00034 {
00035     Q_OBJECT
00036 public:
00037     explicit Darts(QObject *parent = nullptr);
00038     ~Darts();
```

```
00039
00040     QList<Joueur> getListJoueur() const;
00041     int getManche() const;
00042     Darts *getDarts() const;
00043     int getVoleeMax() const;
00044     int getJoueurActif() const;
00045     int getNbVolees() const;
00046     int getPointVolees() const;
00047     int getPremierJoueur() const;
00048     int getDernierJoueur() const;
00049     QString getModeDeJeu() const;
00050     QString testerModeDeJeu();
00051     Solution *getSolution() const;
00052     void setVoleeMax(int voleeMax);
00053     void setManche(int manche);
00054     void receptionnerImpact(int typePoint, int point);
00055     void initialiserPartie(QStringList joueurList, QString modeJeu);
00056     void reinitialiserPartie();
00057     void arreterPartie();
00058     void configurationTournois(QStringList joueurList, QString modeJeu, QString
      nomTournois);
00059     void demarrerTournois();
00060     void receptionnerImpactTournois(int typePoint, int point);
00061
00062 signals:
00063     void miseAJourPoint();
00064     void nouvelleManche();
00065     void nouvelImpact(int,int,int);
00066     void voleeAnnulee();
00067     void finPartie(QString, int, bool);
00068     void etatPartieFini();
00069     void changementJoueurActif();
00070     void miseAJourMoyenneVolee();
00071     void miseAJourMoyenneVoleeTournois();
00072     void afficherNouvellePartie();
00073     void changerEtatPartie();
00074     void actualiserCible();
00075     void jouerSon(QString son);
00076     void afficherTournois(QString modeJeu, QString nomTournois);
00077     void debuterTournois();
00078     void etatPartieTournois();
00079     void miseAJourPointTournois();
00080     void etatPartieAttenteTournois();
00081     void changementJoueurActifTournoi();
00082     void finTournois(QString, QString, QList<Joueur>);
00083     void afficherInfoTournois();
00084
00085 public slots:
00086
00087 private:
00088     Solution *solution;
00089     QList<Joueur> joueurs;
00090     QList<Joueur> joueursTournoisEliminer;
00091     QStringList joueur;
00092     int nbJoueur;
00093     int joueurActif;
00094     int manche;
00095     int pointLancer;
00096     int voleeMax;
00097     int nbVolees;
00098     QString ModeDeJeu;
00099     QString NomTournois;
00100     int pointVoleeEnCours;
00101     int premierJoueur;
00102     int dernierJoueur;
00103
00104     void enleverPointImpact();
00105     void gererManche();
00106     void gererMancheTournois();
00107     void gererVoleeMax();
00108     QString calculerGagnant();
00109     void testerImpact(int typePoint);
00110     void testerImpactTournois(int typePoint);
00111     void testerSiJoueurEliminer();
00112     void controlerJoueurEliminer();
00113     void calculerMoyenneVolees();
00114     void calculerMoyenneVoleesTournois();
00115     void testerNombreJoueurRestand();
00116     void testerPoint(int typePoint, int point);
00117     void calculerPoints(int point, int typePoint);
00118     void gererFinPartieTournois();
00119     bool estDernier();
00120     void initialiserFinTournois();
00121 };
00122
00123 #endif // DARTS_H
```

8.11 Référence du fichier ihm.cpp

Définition de la classe Ihm (Module Ecran-DARTS)

```
#include "ihm.h"
#include "ui_ihm.h"
#include <QTime>
#include <QAction>
#include <QFileInfo>
#include <QPainter>
```

8.11.1 Description détaillée

Définition de la classe Ihm (Module Ecran-DARTS)

Auteur

Bounoir Fabien

Version

0.3

Définition dans le fichier ihm.cpp.

8.12 ihm.cpp

```
00001 #include "ihm.h"
00002 #include "ui_ihm.h"
00003
00004 #include <QTime>
00005 #include <QAction>
00006 #include <QFileInfo>
00007 #include <QPainter>
00008
00026 Ihm::Ihm(QWidget *parent) :
00027     QWidget(parent),
00028     ui(new Ui::Ihm),
00029     musique(qApp->applicationDirPath() + CHEMIN_FICHIER_MUSIQUE + "music.wav",this),
00030     musiquePause(qApp->applicationDirPath() + CHEMIN_FICHIER_MUSIQUE + "pause.wav",
      this),
00031     compteurDureePartie(0),
00032     messageStatut("Volée ")
00033 {
00034     ui->setupUi(this);
00035     qDebug() << Q_FUNC_INFO;
00036
00037     darts = new Darts(this);
00038     communication = new Communication(darts, this);
00039
00040     // Initialiser l'horloge
00041     initialiserHorloge();
00042
00043     // Raccourcis Quitter/ChangerPage (mode debug)
00044     attribuerRaccourcisClavier();
00045
00046     // Démarrer la communication bluetooth
00047     communication->demarrer();
00048
00049     // Initialiser les connexions signal/slot
00050     initialiserEvenements();
00051
00052     //configuration affichage des regles
00053     initialiserAffichageRegle();
00054
00055     //Afficher la page d'attente
00056     afficherNouvellePartie();
00057 }
00058
00064 Ihm::~Ihm()
```

```
00065 {
00066     delete ui;
00067     qDebug() << Q_FUNC_INFO;
00068 }
00069
00075 void Ihm::initialiserEvenements()
00076 {
00077     connect(communication, SIGNAL(appareilConnecter()) , this, SLOT(
      afficherAtteneteConfiguration()));
00078     connect(communication, SIGNAL(afficherAtteneteConnexion()), this,
      SLOT(afficherAtteneteConnexion()));
00079     connect(darts, SIGNAL(afficherNouvellePartie()), this, SLOT(
      afficherPartie()));
00080     connect(communication, SIGNAL(resetPartie()), this, SLOT(
      afficherNouvellePartie()));
00081     connect(darts, SIGNAL(finPartie(QString, int, bool)), this, SLOT(
      finirPartie(QString, int, bool)));
00082     connect(darts, SIGNAL(changementJoueurActif()), this, SLOT(
      mettreAJourJoueur()));
00083     connect(darts, SIGNAL(nouvelImpact(int, int, int)), this, SLOT(
      afficherImpact(int,int)));
00084     connect(darts, SIGNAL(miseAJourPoint()), this, SLOT(mettreAJourScore()));
00085     connect(darts, SIGNAL(miseAJourPointTournois()), this, SLOT(
      mettreAJourScoreTournois()));
00086     connect(darts, SIGNAL(nouvelleManche()), this, SLOT(mettreAJourManche()));
00087     connect(darts, SIGNAL(voleeAnnulee()), this, SLOT(afficherVoleeAnnulee()));
00088     connect(darts, SIGNAL(miseAJourMoyenneVolee()), this, SLOT(
      mettreAJourMoyenneVolee()));
00089     connect(darts, SIGNAL(miseAJourMoyenneVoleeTournois()), this, SLOT(
      mettreAJourMoyenneVoleeTournois()));
00090     connect(darts->getSolution(), SIGNAL(solutionTrouver(QString)), this, SLOT(
      mettreAJoursolution(QString)));
00091     connect(darts, SIGNAL(actualiserCible()), this, SLOT(mettreAJourCible()));
00092     connect(communication, SIGNAL(pause()), this, SLOT(
      mettrePausePartie()));
00093     connect(communication, SIGNAL(play()), this, SLOT(
      relancerpartie()));
00094     connect(communication, SIGNAL(erreurBluetooth(QString)), this, SLOT(
      mettreAJourMessageStatut(QString)));
00095     connect(darts, SIGNAL(jouerSon(QString)), this, SLOT(jouerSon(QString)));
00096     connect(communication, SIGNAL(jouerSon(QString)), this, SLOT(
      jouerSon(QString)));
00097     connect(communication, SIGNAL(afficherRegle(QString)), this, SLOT(
      lancerRegle(QString)));
00098     connect(communication, SIGNAL(stopperRegle()), this, SLOT(
      StopperLectureRegle()));
00099     connect(darts, SIGNAL(afficherTournois(QString, QString)), this, SLOT(
      initialiserAffichageTournois(QString, QString)));
00100     connect(darts, SIGNAL(debuterTournois()), this, SLOT(lancerTournois()));
00101     connect(darts, SIGNAL(changementJoueurActifTournoi()), this, SLOT(
      mettreAJourJoueurTournoi()));
00102     connect(darts, SIGNAL(finTournois(QString,QString,QList<Joueur>)), this, SLOT(
      afficherFinTournois(QString,QString,QList<Joueur>)));
00103     connect(darts, SIGNAL(afficherInfoTournois()), this, SLOT(
      afficherInformationTournois()));
00104
00105 }
00106
00112 void Ihm::attribuerRaccourcisClavier()
00113 {
00114     QAction *quitter = new QAction(this);
00115     quitter->setShortcut(QKeySequence(QKeySequence(Qt::Key_Up)));       //fleche du haut pour quitter
       l'application
00116     addAction(quitter);
00117     connect(quitter, SIGNAL(triggered()), this, SLOT(fermerApplication()));   // Pour
       fermer l'application
00118
00119 #ifndef QT_NO_DEBUG_OUTPUT
00120     QAction *actionAllerDroite = new QAction(this);
00121     actionAllerDroite->setShortcut(QKeySequence(Qt::Key_Right));
00122     addAction(actionAllerDroite);
00123     connect(actionAllerDroite, SIGNAL(triggered()), this, SLOT(allerPageSuivante()));//
       Pour passer à l'écran suivant
00124     QAction *actionAllerGauche = new QAction(this);
00125     actionAllerGauche->setShortcut(QKeySequence(Qt::Key_Left));
00126     addAction(actionAllerGauche);
00127     connect(actionAllerGauche, SIGNAL(triggered()), this, SLOT(
      allerPagePrecedente()));// Pour revenir à l'écran précédent
00128 #endif
00129 }
00130
00136 void Ihm::actualiserHeure()
00137 {
00138     QString affichageHeure;
00139     QTime heure = QTime::currentTime();
00140     affichageHeure = "<font color=\"#6D2B6B\">" + heure.toString("hh : mm ") + "</font>";
00141     ui->labelHeureAttente->setText(affichageHeure);
00142     ui->labelHeureStatistique->setText(affichageHeure);
00143     ui->labelHeureTournois->setText(heure.toString("hh : mm "));
00144 }
00145
```

```
00151 void Ihm::mettreAJourScore()
00152 {
00153     QString score;
00154     int premierJoueurAfficher = 0;
00155     int dernierJoueurAfficher = darts->getListJoueur().size();
00156 
00157     if(darts->getListJoueur().size() > 7)
00158     {
00159         premierJoueurAfficher = darts->getJoueurActif();
00160         dernierJoueurAfficher = darts->getJoueurActif() + 6;
00161 
00162         while(dernierJoueurAfficher > darts->getListJoueur().size() ||
      premierJoueurAfficher < 0)
00163         {
00164             if(premierJoueurAfficher < 0)
00165             {
00166                 premierJoueurAfficher++;
00167                 dernierJoueurAfficher++;
00168             }
00169 
00170             if(dernierJoueurAfficher > darts->getListJoueur().size())
00171             {
00172                 premierJoueurAfficher--;
00173                 dernierJoueurAfficher--;
00174             }
00175         }
00176     }
00177     for(int i = premierJoueurAfficher; i < dernierJoueurAfficher; i++)
00178     {
00179         score += darts->getListJoueur()[i].getNom() + " : " + QString::number(
      darts->getListJoueur()[i].getScore()) + "\n"; // "          " +
00180     }
00181     ui->scoreActuel->setText(score);
00182 }
00183 
00189 void Ihm::mettreAJourManche()
00190 {
00191     ui->manche->setText(QString::number(darts->getManche()));
00192     ui->tournoisManche->setText("Manche " + QString::number(darts->
      getManche()));
00193 }
00194 
00202 void Ihm::afficherImpact(int typePoint, int point)
00203 {
00204     if(QFileInfo(qApp->applicationDirPath() + "/impact/IMPACT_" + QString::number(typePoint) + "_" +
      QString::number(point) + ".png").exists())       //test si l'image existe
00205     {
00206         QImage impact(qApp->applicationDirPath() + "/impact/IMPACT_" + QString::number(typePoint) + "_" +
      QString::number(point) + ".png");
00207 
00208         QPixmap cibleImpacte = ui->labelVisualisationimpact->pixmap()->copy(); // on récupère l'image
       précédente;
00209 
00210         QPainter p(&cibleImpacte);
00211 
00212         p.drawImage(QPoint(0, 0), impact);
00213 
00214         p.end();
00215 
00216         ui->labelVisualisationimpact->setPixmap(cibleImpacte);
00217         ui->ImpactCibleTournois->setPixmap(cibleImpacte);
00218     }
00219 
00220     mettreAJourMessageStatut(typePoint, point);
00221 }
00222 
00230 void Ihm::mettreAJourMessageStatut(int typePoint, int point)
00231 {
00232 
00233     switch(typePoint){
00234         case TRIPLE_POINT:
00235             messageStatut += " T" + QString::number(point);
00236         break;
00237         case DOUBLE_POINT:
00238             messageStatut += " D" + QString::number(point);
00239         break;
00240         case ZERO_POINT:
00241             messageStatut += " " + QString::number(ZERO_POINT);
00242         break;
00243         default:
00244             messageStatut += " " + QString::number(point);
00245         break;
00246     }
00247     ui->labelStatut->setStyleSheet("color: rgb(109, 43,107); border-color: rgb(109, 43,107);");
00248     ui->labelStatut->setText(messageStatut + "     " +QString::number(
      darts->getPointVolees()) + " Point(s)");
00249     ui->statutImpactTournois->setText(messageStatut + "     " +QString::number(
      darts->getPointVolees()) + " Point(s)");
00250 }
00251 
00252 
```

```
00258 void Ihm::mettreAJourJoueur()
00259 {
00260     QString nomjoueur;
00261
00262     int premierJoueurAfficher = 0;
00263     int dernierJoueurAfficher = darts->getListJoueur().size();
00264
00265     if(darts->getListJoueur().size() > 7)
00266     {
00267         premierJoueurAfficher = darts->getJoueurActif();
00268         dernierJoueurAfficher = darts->getJoueurActif() + 6;
00269
00270         while(dernierJoueurAfficher > darts->getListJoueur().size() ||
      premierJoueurAfficher < 0)
00271         {
00272             if(premierJoueurAfficher < 0)
00273             {
00274                 premierJoueurAfficher++;
00275                 dernierJoueurAfficher++;
00276             }
00277
00278             if(dernierJoueurAfficher > darts->getListJoueur().size())
00279             {
00280                 premierJoueurAfficher--;
00281                 dernierJoueurAfficher--;
00282             }
00283         }
00284     }
00285
00286     for(int i = premierJoueurAfficher; i < dernierJoueurAfficher; i++)
00287     {
00288         if(i == darts->getJoueurActif())    // test si le joueur est le joueur qui doit
       jouer
00289         {
00290             nomjoueur += "    " + darts->getListJoueur()[i].getNom() + "\n";  //joueur
       joue
00291         }
00292         else
00293         {
00294             nomjoueur += "            " + darts->getListJoueur()[i].getNom() + "\n";
      //joueur en attente de son tour //
00295         }
00296     }
00297     ui->nomJoueur->setText(nomjoueur);
00298 }
00299
00305 void Ihm::mettreAJourMoyenneVolee()
00306 {
00307     QString moyenneVoleeJoueur;
00308     QString moyenneVoleesJoueur;
00309
00310     int premierJoueurAfficher = 0;
00311     int dernierJoueurAfficher = darts->getListJoueur().size();
00312
00313     if(darts->getListJoueur().size() > 7)
00314     {
00315         for(int i = premierJoueurAfficher; i < 7; i++)
00316         {
00317             moyenneVoleeJoueur += darts->getListJoueur()[i].getNom() + " : " +
      QString::number(darts->getListJoueur()[i].getMoyenneVolee()) +" \n"; //"         " +
00318         }
00319
00320         for(int i = 7; i < dernierJoueurAfficher; i++)
00321         {
00322             moyenneVoleesJoueur += darts->getListJoueur()[i].getNom() + " : " +
      QString::number(darts->getListJoueur()[i].getMoyenneVolee()) +" \n"; //"         " +
00323         }
00324
00325         ui->moyenneVolee->setText(moyenneVoleeJoueur);
00326         ui->lineSeparateurMoyenne->setVisible(true);
00327         ui->moyenneVolee2->setText(moyenneVoleesJoueur);
00328     }
00329     else
00330     {
00331         for(int i = premierJoueurAfficher; i < dernierJoueurAfficher; i++)
00332         {
00333             moyenneVoleeJoueur += darts->getListJoueur()[i].getNom() + " : " +
      QString::number(darts->getListJoueur()[i].getMoyenneVolee()) +" \n"; //"         " +
00334         }
00335         ui->moyenneVolee->setText(moyenneVoleeJoueur);
00336     }
00337     ui->labelMoyenneVolees->setVisible(true);
00338     ui->lineScoreActuel->setVisible(true);
00339     ui->labelMoyenneVoleesStatistique->setVisible(true);
00340     ui->moyenneVolees->setText(moyenneVoleeJoueur + moyenneVoleesJoueur);
00341 }
00342
00348 void Ihm::afficherPartie()
00349 {
00350     musique.stop();
00351
```

```
00352     ui->typeJeu->setText(darts->getModeDeJeu());
00353
00354     mettreAJourJoueur();
00355     compteurDureePartie = 0;
00356     connect(timerHorloge, SIGNAL(timeout()),this,SLOT(
      afficherDureePartie())); // Pour le comptage et l'affichage de la durée d'une séance
00357
00358     allerPage(Ihm::PageJeu);
00359
00360     mettreAJourScore();
00361 }
00362
00368 void Ihm::afficherVoleeAnnulee()
00369 {
00370     ui->labelStatut->setText("Volée annulée !");
00371     ui->statutImpactTournois->setText("Volée annulée !");
00372 }
00373
00381 void Ihm::finirPartie(QString gagnant, int voleeMaxJoueur, bool tournois)
00382 {
00383     player->stop();
00384     musique.play();
00385
00386     if(tournois == true)
00387     {
00388         ui->labelMoyenneVoleesStatistique->setVisible(true);
00389         ui->moyenneVolees->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom() + "  " + QString::number(
      darts->getListJoueur()[darts->getPremierJoueur()].getMoyenneVolee())
       + "\n" + darts->getListJoueur()[darts->getDernierJoueur()].getNom()
      + "  " +QString::number(darts->getListJoueur()[darts->
      getDernierJoueur()].getMoyenneVolee()));
00390         ui->moyenneVolees->setVisible(true);
00391     }
00392
00393     disconnect(timerHorloge, SIGNAL(timeout()), this, SLOT(
      afficherDureePartie())); // Pour le comptage et l'affichage de la durée d'une séance
00394     ui->winnerPartie->setText(gagnant);
00395     ui->voleeMax->setText(QString::number(voleeMaxJoueur) + " points");
00396     ui->nbVolees->setText(QString::number(darts->getNbVolees()));
00397     //communication->miseAJourEtatPartieFin();
00398     allerPage(Ihm::PageStatistique);
00399 }
00400
00406 void Ihm::afficherNouvellePartie()
00407 {
00408      player->stop();
00409      allerPage(Ihm::PageAttente);
00410      ui->manche->setText("1");
00411      ui->nomJoueur->setText("");
00412      ui->scoreActuel->setText("");
00413      ui->typeJeu->setText("");
00414      ui->winnerPartie->setText("Winner ....");
00415      ui->labelStatut->setText("");
00416      ui->moyenneVolee->setText("");
00417      ui->moyenneVolee2->setText("");
00418      ui->nbVolees->setText("");
00419      ui->voleeMax->setText("");
00420      ui->moyenneVolees->setText("");
00421      ui->labelMoyenneVolees->setVisible(false);
00422      ui->lineScoreActuel->setVisible(false);
00423      ui->lineSeparateurMoyenne->setVisible(false);
00424      ui->labelMoyenneVoleesStatistique->setVisible(false);
00425      ui->ecranPartie->setStyleSheet("
      QWidget#ecranPartie{background-image:url(:/ressources/background.jpg);}");
00426      mettreAJourCible();
00427      ui->labelINfoMatch->setVisible(false);
00428
00429      //ecran Tournois
00430      ui->tournoisNom->setText("----------");
00431      ui->modeDeJeuTournois->setText("");
00432      ui->nomJoueurTournois1->setText("");
00433      ui->nomJoueurTournois2->setText("");
00434      ui->scoreJoueurTournois1->setText("\n\n");
00435      ui->scoreJoueurTournois2->setText("\n\n");
00436      ui->moyenneJoueurTournois1->setText("\n\n" );
00437      ui->moyenneJoueurTournois2->setText("\n\n" );
00438      ui->tournoisManche->setText("Manche 1");
00439      ui->statutImpactTournois->setText("");
00440
00441
00442      // configurer la musique
00443      musique.setLoops(QSound::Infinite);
00444      musiquePause.setLoops(QSound::Infinite);
00445      musiquePause.stop();
00446      // jouer la musique
00447      musique.play();
00448 }
00449
00455 void Ihm::allerPageSuivante()
00456 {
```

```
00457     int ecranCourant = Page(ui->ecranDarts->currentIndex());
00458     int ecranSuivant = (ecranCourant+1)%int(Ihm::NbPages);
00459     ui->ecranDarts->setCurrentIndex(ecranSuivant);
00460 }
00461
00468 void Ihm::allerPage(Ihm::Page page)
00469 {
00470     ui->ecranDarts->setCurrentIndex(page);
00471 }
00472
00478 void Ihm::allerPagePrecedente()
00479 {
00480     int ecranCourant = ui->ecranDarts->currentIndex();
00481     int ecranPrecedent = (ecranCourant-1)%int(Ihm::NbPages);
00482     if(ecranPrecedent == -1)
00483         ecranPrecedent = NbPages-1;
00484     ui->ecranDarts->setCurrentIndex(ecranPrecedent);
00485 }
00486
00492 void Ihm::fermerApplication()
00493 {
00494     this->close();
00495 }
00496
00502 void Ihm::afficherAttenteConfiguration()
00503 {
00504     ui->labelStatutAttente->setText(" Attente configuration de la partie ");
00505 }
00506
00512 void Ihm::afficherAttenteConnexion()
00513 {
00514     ui->labelStatutAttente->setText(" En attente de connexion ");
00515 }
00516
00522 void Ihm::afficherPretLancerTournois()
00523 {
00524     ui->labelStatutAttente->setText(" Prêt à lancer le tournoi ");
00525 }
00526
00533 void Ihm::afficherDureePartie()
00534 {
00535     QTime duree(0, 0);
00536     compteurDureePartie++;
00537     QTime dureeSeance = duree.addSecs(compteurDureePartie);
00538     if(compteurDureePartie >= 3600)
00539     {
00540         ui->ecranPartie->setStyleSheet("
      QWidget#ecranPartie{background-image:url(:/ressources/backgroundHeure.jpg);}");
00541         ui->labelTempsPartie->setText(dureeSeance.toString("hh : mm : ss"));
00542         ui->tempsPartie->setText(dureeSeance.toString("hh : mm : ss"));
00543         ui->tournoisChrono->setText(dureeSeance.toString("hh : mm : ss"));
00544     }
00545     else
00546     {
00547         ui->labelTempsPartie->setText(dureeSeance.toString("mm : ss"));
00548         ui->tempsPartie->setText(dureeSeance.toString("mm : ss"));
00549         ui->tournoisChrono->setText(dureeSeance.toString("mm : ss"));
00550     }
00551 }
00552
00559 void Ihm::mettreAJoursolution(QString solution)
00560 {
00561     ui->labelStatut->setStyleSheet("color: rgb(179, 0,5); border-color: rgb(179, 0,5);");
00562     ui->labelStatut->setText(solution);
00563     ui->statutImpactTournois->setText(solution);
00564 }
00565
00571 void Ihm::mettrePausePartie()
00572 {
00573     disconnect(timerHorloge, SIGNAL(timeout()),this,SLOT(
      afficherDureePartie())); // mettre en pause le chronometrage de la partie
00574     sauvegardeImpactEncours = ui->labelVisualisationimpact->pixmap()->copy();
00575     QImage pause(":pause.png");
00576     QPixmap cibleImpacte = ui->labelVisualisationimpact->pixmap()->copy(); // on récupère l'image
       précédente;
00577     QPainter p(&cibleImpacte);
00578     p.drawImage(QPoint(0, 0), pause);
00579     p.end();
00580     ui->labelVisualisationimpact->setPixmap(cibleImpacte);
00581     ui->ImpactCibleTournois->setPixmap(cibleImpacte);
00582     ui->labelTempsPartie->setStyleSheet("color: rgb(179, 0,5);");
00583
00584     musiquePause.play();
00585 }
00586
00592 void Ihm::relancerpartie()
00593 {
00594     ui->labelVisualisationimpact->setPixmap(sauvegardeImpactEncours);
00595     ui->ImpactCibleTournois->setPixmap(sauvegardeImpactEncours);
00596     ui->labelTempsPartie->setStyleSheet("color: rgb(109, 43,107);");
00597     connect(timerHorloge, SIGNAL(timeout()),this,SLOT(
```

```
    afficherDureePartie())); // relancer le chronometrage de la partie
00598     qDebug() << Q_FUNC_INFO << endl;
00599     musiquePause.stop();
00600 }
00601
00607 void Ihm::initialiserHorloge()
00608 {
00609     timerHorloge = new QTimer(this); // Instancie dynamiquement le temporisateur du
     rafraichissement de l'heure
00610     connect(timerHorloge, SIGNAL(timeout()),this,SLOT(
    actualiserHeure())); // Pour le déclenchement périodique de l'affichage de l'heure
00611     timerHorloge->start(PERIODE_HORLOGE);  // Toutes les secondes (1000 ms)
00612 }
00613
00619 void Ihm::mettreAJourCible()
00620 {
00621     ui->labelVisualisationimpact->setPixmap(QPixmap(":/ressources/cible.png"));
00622     ui->ImpactCibleTournois->setPixmap(QPixmap(":/ressources/cible.png"));
00623     sauvegardeImpactEncours = ui->labelVisualisationimpact->pixmap()->copy();
00624     messageStatut = "Volée ";
00625 }
00626
00633 void Ihm::mettreAJourMessageStatut(QString statut)
00634 {
00635     ui->labelStatutAttente->setText(statut);
00636 }
00637
00638
00645 void Ihm::jouerSon(QString son)
00646 {
00647     QSound::play(qApp->applicationDirPath() + CHEMIN_FICHIER_MUSIQUE + son);
00648     if(!QFileInfo(qApp->applicationDirPath() + CHEMIN_FICHIER_MUSIQUE + son).exists()
    )
00649     {
00650         qDebug() << Q_FUNC_INFO << "Pour avoir les sons, ajouter le pack disponible à cette adresse dans le
     build de votre application:"<<endl;
00651         qDebug() << Q_FUNC_INFO << "
    https://drive.google.com/file/d/1vH0tLe8lsu2VQLISDL94nJBA2arfLcbG/view?usp=sharing"<<endl;
00652     }
00653 }
00654
00660 void Ihm::initialiserAffichageRegle()
00661 {
00662     player = new QMediaPlayer;
00663
00664     videoWidget = new QVideoWidget(this);
00665     ui->verticalLayoutRegle->addWidget(videoWidget);
00666     player->setVideoOutput(videoWidget);
00667
00668     connect(player, SIGNAL(stateChanged(QMediaPlayer::State)), this, SLOT(
    stateChanged(QMediaPlayer::State)));
00669     connect(player, SIGNAL(error(QMediaPlayer::Error)), this, SLOT(
    error(QMediaPlayer::Error)));
00670 }
00671
00678 void Ihm::lancerRegle(QString regle)
00679 {
00680     sauverEtatPartie = communication->getEtatPartie();
00681     communication->miseAJourEtatPartieRegle();
00682
00683     player->setMedia(QUrl::fromLocalFile(QCoreApplication::applicationDirPath() + QString("/" + regle
     + ".mp4")));
00684
00685     musiquePause.stop();
00686     musique.stop();
00687     allerPage(Ihm::PageRegle);
00688     player->play();
00689 }
00690
00696 void Ihm::testerEtatPartie()
00697 {
00698     if(sauverEtatPartie == 0)
00699     {
00700         communication->miseAJourEtatPartieAttente();
00701         musique.play();
00702         allerPage(Ihm::PageAttente);
00703     }
00704     else if(sauverEtatPartie == 1)
00705     {
00706         communication->miseAJourEtatPartieEnCours();
00707         allerPage(Ihm::PageJeu);
00708         relancerpartie();
00709     }
00710     else if(sauverEtatPartie == 2)
00711     {
00712         communication->miseAJourEtatPartieFin();
00713         musique.play();
00714         allerPage(Ihm::PageStatistique);
00715     }
00716     else if(sauverEtatPartie == 3)
00717     {
```

```
00718         communication->miseAJourEtatPartiePause();
00719         musiquePause.play();
00720         allerPage(Ihm::PageJeu);
00721     }
00722 }
00723
00730 void Ihm::stateChanged(QMediaPlayer::State state)
00731 {
00732     qDebug() << Q_FUNC_INFO << state;
00733     if(state == QMediaPlayer::StoppedState)
00734     {
00735         testerEtatPartie();
00736     }
00737 }
00738
00744 void Ihm::StopperLectureRegle()
00745 {
00746     player->stop();
00747 }
00748
00755 void Ihm::error(QMediaPlayer::Error error)
00756 {
00757     qDebug() << Q_FUNC_INFO << player->errorString() << error;
00758     testerEtatPartie();
00759 }
00760
00766 void Ihm::initialiserAffichageTournois(QString modeJeu, QString
      nomTournois)
00767 {
00768     ui->tournoisNom->setText("Tournoi " + nomTournois);
00769     ui->modeDeJeuTournois->setText(modeJeu);
00770
00771     ui->nomJoueurTournois1->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom());
00772     ui->nomJoueurTournois2->setText(darts->getListJoueur()[
      darts->getDernierJoueur()].getNom());
00773
00774     ui->scoreJoueurTournois1->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom() + "\n\n" + QString::number(
      darts->getListJoueur()[darts->getPremierJoueur()].getScore()));
00775     ui->scoreJoueurTournois2->setText(darts->getListJoueur()[
      darts->getDernierJoueur()].getNom() + "\n\n" + QString::number(
      darts->getListJoueur()[darts->getDernierJoueur()].getScore()));
00776
00777     ui->moyenneJoueurTournois1->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom() + "\n\n0" );
00778     ui->moyenneJoueurTournois2->setText(darts->getListJoueur()[
      darts->getDernierJoueur()].getNom() + "\n\n0" );
00779
00780     ui->tournoisManche->setText("Manche " + QString::number(darts->
      getManche()));
00781
00782     afficherPretLancerTournois();
00783 }
00784
00790 void Ihm::lancerTournois()
00791 {
00792     musique.stop();
00793
00794     compteurDureePartie = 0;
00795     connect(timerHorloge, SIGNAL(timeout()),this,SLOT(
      afficherDureePartie())); // Pour le comptage et l'affichage de la durée d'une séance
00796
00797     allerPage(Ihm::PageTournois);
00798 }
00799
00805 void Ihm::mettreAJourScoreTournois()
00806 {
00807     ui->scoreJoueurTournois1->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom() + "\n\n" + QString::number(
      darts->getListJoueur()[darts->getPremierJoueur()].getScore()));
00808     ui->scoreJoueurTournois2->setText(darts->getListJoueur()[
      darts->getDernierJoueur()].getNom() + "\n\n" + QString::number(
      darts->getListJoueur()[darts->getDernierJoueur()].getScore()));
00809 }
00810
00816 void Ihm::mettreAJourMoyenneVoleeTournois()
00817 {
00818     ui->moyenneJoueurTournois1->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom() + "\n\n" + QString::number(
      darts->getListJoueur()[darts->getPremierJoueur()].getMoyenneVolee())
      );
00819     ui->moyenneJoueurTournois2->setText(darts->getListJoueur()[
      darts->getDernierJoueur()].getNom() + "\n\n" +QString::number(
      darts->getListJoueur()[darts->getDernierJoueur()].getMoyenneVolee())
      );
00820 }
00821
00827 void Ihm::mettreAJourJoueurTournoi()
00828 {
00829     if(darts->getJoueurActif() == darts->
```

```
      getPremierJoueur())
00830     {
00831         ui->nomJoueurTournois1->setText(" " + darts->getListJoueur()[
      darts->getPremierJoueur()].getNom() + " ");
00832         ui->nomJoueurTournois2->setText(darts->getListJoueur()[
      darts->getDernierJoueur()].getNom());
00833     }
00834     else
00835     {
00836         ui->nomJoueurTournois1->setText(darts->getListJoueur()[
      darts->getPremierJoueur()].getNom());
00837         ui->nomJoueurTournois2->setText(" " + darts->getListJoueur()[
      darts->getDernierJoueur()].getNom() + " ");
00838     }
00839 }
00840
00846 void Ihm::afficherFinTournois(QString nomGagnant,QString nomTournois,QList<Joueur>
      joueurs)
00847 {
00848     QString joueurTournois = "";
00849
00850     for(int i = 0 ; i < joueurs.size(); i++)
00851     {
00852
00853         if(i == 0)
00854         {
00855             joueurTournois = "          1er  " + joueurs[i].getNom() + "\n";
00856         }
00857         else
00858         {
00859             joueurTournois = joueurTournois +"          "+ QString::number(i + 1) + "ème  " + joueurs[i].
      getNom() + "\n";
00860         }
00861     }
00862
00863     ui->winnerTournois->setText(" " + nomGagnant + " grand(e) gagnant(e) du tournoi " + nomTournois +"
      ");
00864     ui->recapPlaceTournois->setText(joueurTournois);
00865     allerPage(Ihm::PageFinTournois);
00866 }
00867
00873 void Ihm::afficherInformationTournois()
00874 {
00875     ui->labelINfoMatch->setVisible(true);
00876 }
```

8.13 Référence du fichier ihm.h

Déclaration de la classe Ihm (Module Ecran-DARTS)

```
#include "communication.h"
#include "darts.h"
#include <QWidget>
#include <QTimer>
#include <QSound>
#include <QtMultimedia>
#include <QtMultimediaWidgets>
```

Classes

— class Ihm
Déclaration de la classe Ihm (Module Ecran-DARTS)

Espaces de nommage

— Ui

Macros

- #define CHEMIN_FICHIER_MUSIQUE "/son/"
  Définit le chemin pour les sons.
- #define PERIODE_HORLOGE 1000
  Définit la périodicité de l'horloge en millisecondes.

8.13.1 Description détaillée

Déclaration de la classe Ihm (Module Ecran-DARTS)

Version

0.3

Auteur

Bounoir Fabien

Définition dans le fichier ihm.h.

8.13.2 Documentation des macros

8.13.2.1 CHEMIN_FICHIER_MUSIQUE

```
#define CHEMIN_FICHIER_MUSIQUE "/son/"
```

Définit le chemin pour les sons.

Définition à la ligne 32 du fichier ihm.h.

Référencé par Ihm : :jouerSon().

8.13.2.2 PERIODE_HORLOGE

```
#define PERIODE_HORLOGE 1000
```

Définit la périodicité de l'horloge en millisecondes.

Définition à la ligne 26 du fichier ihm.h.

Référencé par Ihm : :initialiserHorloge().

8.14 ihm.h

```
00001 #ifndef IHM_H
00002 #define IHM_H
00003
00013 #include "communication.h"
00014 #include "darts.h"
00015 #include <QWidget>
00016 #include <QTimer>
00017 #include <QSound>
00018
00019 #include <QtMultimedia>
00020 #include <QtMultimediaWidgets>
00021
00026 #define PERIODE_HORLOGE     1000
00027
00032 #define CHEMIN_FICHIER_MUSIQUE "/son/"
00033
00034 namespace Ui {
00035 class Ihm;
00036 }
00037
00043 class Ihm : public QWidget
00044 {
00045     Q_OBJECT
00046
00047 public:
00048     explicit Ihm(QWidget *parent = nullptr);
00049     ~Ihm();
00050
00051 private:
00052     Ui::Ihm *ui;
00053     QTimer *timerHorloge;
00054     Communication *communication;
00055     Darts *darts;
00056     QSound musique;
00057     QSound musiquePause;
00058     int compteurDureePartie;
00059     int sauverEtatPartie;
00060     QPixmap sauvegardeImpactEncours;
00061     QString messageStatut;
00062
00063     //objet pour l'affichage video
00064     QMediaPlayer *player;
00065     QVideoWidget *videoWidget;
00066
00072     enum Page
00073     {
00074         PageAttente = 0,
00075         PageRegle,
00076         PageJeu,
00077         PageStatistique,
00078         PageTournois,
00079         PageFinTournois,
00080         NbPages
00081     };
00082
00083     void attribuerRaccourcisClavier();
00084     void initialiserEvenements();
00085     void initialiserHorloge();
00086     void mettreAJourMessageStatut(int typePoint, int point);
00087     void initialiserAffichageRegle();
00088     void testerEtatPartie();
00089
00090 public slots:
00091     void actualiserHeure();
00092     void allerPage(Ihm::Page page);
00093     void allerPagePrecedente();
00094     void allerPageSuivante();
00095     void fermerApplication();
00096     void afficherAttenteConfiguration();
00097     void afficherAttenteConnexion();
00098     void afficherPretLancerTournois();
00099     void afficherImpact(int typePoint, int point);
00100     void afficherPartie();
00101     void mettreAJourScore();
00102     void mettreAJourManche();
00103     void mettreAJourMoyenneVolee();
00104     void mettreAJourMoyenneVoleeTournois();
00105     void afficherVoleeAnnulee();
00106     void afficherNouvellePartie();
00107     void finirPartie(QString gagnant, int voleeMaxJoueur, bool tournois);
00108     void mettreAJourJoueur();
00109     void afficherDureePartie();
00110     void mettreAJoursolution(QString solution);
00111     void mettrePausePartie();
00112     void relancerpartie();
00113     void mettreAJourCible();
00114     void mettreAJourMessageStatut(QString);
00115     void jouerSon(QString son);
```

```
00116     void lancerRegle(QString regle);
00117     void stateChanged(QMediaPlayer::State state);
00118     void error(QMediaPlayer::Error error);
00119     void StopperLectureRegle();
00120     void initialiserAffichageTournois(QString modeJeu, QString nomTournois);
00121     void lancerTournois();
00122     void mettreAJourScoreTournois();
00123     void mettreAJourJoueurTournoi();
00124     void afficherFinTournois(QString nomGagnant, QString nomTournois, QList<Joueur> joueurs);
00125     void afficherInformationTournois();
00126 };
00127
00128 #endif // IHM_H
```

8.15 Référence du fichier joueur.cpp

Définition de la classe Joueur (Module Ecran-DARTS)

```
#include "joueur.h"
#include <QtDebug>
```

8.15.1 Description détaillée

Définition de la classe Joueur (Module Ecran-DARTS)

Auteur

Bounoir Fabien

Version

0.3

Définition dans le fichier joueur.cpp.

8.16 joueur.cpp

```
00001 #include "joueur.h"
00002
00003 #include <QtDebug>
00004
00024 Joueur::Joueur(QString nom, int score , int nbFlechette): nom(nom), score(score),
      moyenneVolee(0), scoreManchePrecedente(score), nbFlechette(nbFlechette), elimine(false)
00025 {
00026     qDebug() << Q_FUNC_INFO << nom << " " << score << " " << nbFlechette;
00027 }
00028
00035 QString Joueur::getNom() const
00036 {
00037     return this->nom;
00038 }
00039
00046 int Joueur::getScore() const
00047 {
00048     return this->score;
00049 }
00050
00057 int Joueur::getScoreManchePrecedente() const
00058 {
00059     return this->scoreManchePrecedente;
00060 }
00061
00068 int Joueur::getFlechette() const
00069 {
00070     return this->nbFlechette;
00071 }
00072
00079 int Joueur::getMoyenneVolee() const
00080 {
```

```
00081     return this->moyenneVolee;
00082 }
00083
00090 QVector<float> Joueur::getHistoriqueVolees() const
00091 {
00092     return this->historiqueVolees;
00093 }
00094
00101 bool Joueur::getEliminer() const
00102 {
00103     return this->elimine;
00104 }
00105
00112 void Joueur::setEliminer(bool elimine)
00113 {
00114     this->elimine = elimine;
00115 }
00116
00123 void Joueur::setMoyenneVolee(int moyenneVolee)
00124 {
00125     this->moyenneVolee = moyenneVolee;
00126 }
00127
00128
00135 void Joueur::setScore(int score)
00136 {
00137     this->score = score;
00138 }
00139
00146 void Joueur::setScoreManchePrecedente(int
      scoreManchePrecedente)
00147 {
00148     this->scoreManchePrecedente = scoreManchePrecedente;
00149 }
00150
00157 void Joueur::setNbFlechette(int nbFlechette)
00158 {
00159     this->nbFlechette = nbFlechette;
00160 }
00161
00168 void Joueur::addHistoriqueVolees(float volee)
00169 {
00170     historiqueVolees.push_back(volee);
00171 }
```

8.17 Référence du fichier joueur.h

Déclaration de la classe Joueur (Module Ecran-DARTS)

```
#include <QString>
#include <QVector>
```

Classes

— class Joueur
Déclaration de la classe Joueur (Module Ecran-DARTS)

8.17.1 Description détaillée

Déclaration de la classe Joueur (Module Ecran-DARTS)

Version

0.3

Auteur

Bounoir Fabien

Définition dans le fichier joueur.h.

8.18 joueur.h

```
00001 #ifndef JOUEUR_H
00002 #define JOUEUR_H
00003
00004 #include <QString>
00005 #include <QVector>
00006
00021 class Joueur
00022 {
00023 public:
00024     Joueur(QString nom, int score , int nbFlechette);
00025
00026     QString getNom() const;
00027     int getScore() const;
00028     int getScoreManchePrecedente() const;
00029     int getFlechette() const;
00030     int getMoyenneVolee() const;
00031     QVector<float> getHistoriqueVolees() const;
00032     bool getEliminer() const;
00033     void setEliminer(bool elimine);
00034     void setMoyenneVolee(int moyenneVolee);
00035     void setScore(int score);
00036     void setScoreManchePrecedente(int
      scoreManchePrecedente);
00037     void setNbFlechette(int nbFlechette);
00038     void addHistoriqueVolees(float volee);
00039
00040 private:
00041     QString nom;
00042     QVector<float> historiqueVolees;
00043     int score;
00044     int moyenneVolee;
00045     int scoreManchePrecedente;
00046     int nbFlechette;
00047     bool elimine;
00048 };
00049
00050 #endif // JOUEUR_H
```

8.19 Référence du fichier main.cpp

Programme principal ecran-DARTS.

```
#include "ihm.h"
#include <QApplication>
```

Fonctions

— int main (int argc, char ∗argv[ ])

8.19.1 Description détaillée

Programme principal ecran-DARTS.

Crée et affiche la fenêtre principale de l'application ecran-DARTS

Auteur

Bounoir Fabien

Version

0.3

Définition dans le fichier main.cpp.

8.19.2 Documentation des fonctions

8.19.2.1 main()

```
main (
            int argc,
            char * argv[] )
```

Paramètres

argc	
argv[]	

Renvoie

int

Définition à la ligne 21 du fichier main.cpp.

```
00022 {
00023     QApplication a(argc, argv);
00024     Ihm w;
00025     w.showFullScreen();
00026
00027     return a.exec();
00028 }
```

8.20 main.cpp

```
00001 #include "ihm.h"
00002 #include <QApplication>
00003
00021 int main(int argc, char *argv[])
00022 {
00023     QApplication a(argc, argv);
00024     Ihm w;
00025     w.showFullScreen();
00026
00027     return a.exec();
00028 }
00029
00030 // Pour la documentation Doxygen
00031
```

8.21 Référence du fichier README.md

8.22 README.md

```
00001 \mainpage Le projet
00002
00003 \tableofcontents
00004
00005 Le système **DARTS** est un système numérique permettant de jouer au jeu de fléchettes électroniques.
00006
00007 \section section_tdm Table des matières
00008 - \ref page_README
00009 - \ref page_changelog
00010 - \ref page_about
00011 - \ref page_licence
00012
00013 \section section_infos Informations
00014
```

```
00015 \author Fabien Bounoir <bounoirfabien@gmail.com>
00016 \date 2020
00017 \version 0.2
00018 \see https://svn.riouxsvn.com/darts-2020/
00019
00020
00021 \page page_README README
00022
00023 [TOC]
00024
00025 # Projet {#projet}
00026
00027 ## Présentation {#presentation}
00028
00029 Le système **DARTS** est un système numérique permettant de jouer au jeu de fléchettes électroniques.
00030
00031 Le système DARTS est décomposé en trois modules, dont deux modules sont réalisés par des étudiants IR
       :
00032
00033 * Module de gestion de partie (**Mobile-DARTS**) : les joueurs paramètrent et lancent la partie à
       partir d'une application sur un terminal mobile (sous Android) ;
00034
00035 * Module de détection des impacts (Cible-DARTS) : la cible est équipée de capteurs permettant
       d'identifier la zone impactée par les fléchettes envoyées par les joueurs ;
00036
00037 * Module de visualisation de partie (**Écran-DARTS**)  : les joueurs, les arbitres et le public
       peuvent visualiser en "temps réel" le déroulement de la partie (nombre de manche, point restant dans la manche,
       moyenne des volées, ...) sur un écran de télévision.
00038
00039 ## Module de visualisation de partie (Écran-DARTS) {#ecran}
00040
00041 Ce module correspond à la partie "affichage" du système. Il a pour objectifs de réaliser la
       récupération d'informations envoyées par le terminal mobile, le calcul et l'affichage les statistiques pour la
       partie actuelle. Il communique en Bluetooth uniquement avec le terminal mobile Android.
00042
00043 Sur l'écran, les joueurs pourront visualiser en continu :
00044
00045 * le nom des joueurs (si existant), la durée écoulée de la partie ;
00046 * le type de jeu en cours, le score et le nombre de manches gagnées par chaque joueur
00047 * la plus haute et la moyenne des volées de chaque joueur
00048
00049 Les données visualisées sont donc :
00050
00051 * Le type de jeu
00052 * Le numéro de la manche
00053 * Le score de la partie en cours
00054 * La moyenne des volées
00055
00056 ## Informations {#informations}
00057
00058 \author Fabien Bounoir <bounoirfabien@gmail.com>
00059 \date 2020
00060 \version 0.2
00061 \see https://svn.riouxsvn.com/darts-2020/
00062
00063
00064 \page page_about A propos
00065
00066 \author Fabien Bounoir <bounoirfabien@gmail.com>
00067 \date 2020
00068 \version 0.2
00069 \see https://svn.riouxsvn.com/darts-2020/
00070
00071
00072 \page page_licence Licence GPL
00073
00074 This program is free software; you can redistribute it and/or modify
00075 it under the terms of the GNU General Public License as published by
00076 the Free Software Foundation; either version 2 of the License, or
00077 (at your option) any later version.
00078
00079 This program is distributed in the hope that it will be useful,
00080 but WITHOUT ANY WARRANTY; without even the implied warranty of
00081 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00082 GNU General Public License for more details.
00083
00084 You should have received a copy of the GNU General Public License
00085 along with this program; if not, write to the Free Software
00086 Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
```

8.23 Référence du fichier solution.cpp

Définition de la classe Solution (Module Ecran-DARTS)

```
#include "solution.h"
```

8.23.1 Description détaillée

Définition de la classe Solution (Module Ecran-DARTS)

Auteur

Bounoir Fabien

Version

0.3

Définition dans le fichier solution.cpp.

8.24 solution.cpp

```
00001 #include "solution.h"
00002
00020 Solution::Solution(QObject *parent) : QObject(parent), solution("")
00021 {
00022    qDebug() << Q_FUNC_INFO;
00023 }
00024
00030 Solution::~Solution()
00031 {
00032    qDebug() << Q_FUNC_INFO;
00033 }
00034
00035
00042 void Solution::transmettreSolution(int score)
00043 {
00044     qDebug() << Q_FUNC_INFO << "Score = " << score << " : " << solution;
00045     emit solutionTrouver("  " + QString::number(score) + "  " + solution + "  ");
00046 }
00047
00056 bool Solution::aTriple(int points, const int score)
00057 {
00058     if((score / (points*3)) >= 1)
00059     {
00060         return true;
00061     }
00062     return false;
00063 }
00064
00074 bool Solution::rechercherTriple(int &score, QString &combinaison, int start=20)
00075 {
00076     bool trouve = false;
00077     for(int i=start; i>1 && !trouve; i--)
00078     {
00079         if(aTriple(i, score))
00080         {
00081             combinaison = "T" + QString::number(i);
00082             score -= (i*3);
00083             trouve = true;
00084         }
00085     }
00086     return trouve;
00087 }
00088
00097 bool Solution::aDouble(int points, const int score)
00098 {
00099     if((score / (points*2)) >= 1)
00100     {
00101         return true;
00102     }
00103     return false;
00104 }
00105
00114 bool Solution::rechercherDouble(int &score, QString &combinaison)
00115 {
00116     bool trouve = false;
00117     // Remarque : le 2 est plus facile que le D1 !
00118     for(int i=BULL; i>1 && !trouve; i--)
00119     {
00120         // cibles inexistantes ?
00121         if(i > 20)
00122             continue;
00123         if(aDouble(i, score))
00124         {
```

```
00125             combinaison = "D" + QString::number(i);
00126             score -= (i*2);
00127             trouve = true;
00128         }
00129     }
00130     return trouve;
00131 }
00132
00141 bool Solution::aSimple(int points, const int score)
00142 {
00143     if((score / points) >= 1)
00144         return true;
00145     return false;
00146 }
00147
00156 bool Solution::rechercherSimple(int &score, QString &combinaison)
00157 {
00158     bool trouve = false;
00159
00160     for(int i=BULL; i>0 && !trouve; i--)
00161     {
00162         // cibles inexistantes ?
00163         if(i > 20)
00164             continue;
00165         if(aSimple(i, score))
00166         {
00167             combinaison = QString::number(i);
00168             score -= i;
00169             trouve = true;
00170         }
00171     }
00172     return trouve;
00173 }
00174
00183 bool Solution::estDouble(int points, const int score)
00184 {
00185     if(score == (points*2))
00186         return true;
00187     return false;
00188 }
00189
00198 bool Solution::extraireDouble(int &score, int cible)
00199 {
00200     if(aDouble(cible, score))
00201     {
00202         score -= (cible*2);
00203         return true;
00204     }
00205     return false;
00206 }
00207
00217 bool Solution::rechercher(int score, int nbFlechettes, bool still)
00218 {
00219     QString combinaison;
00220     int s = score;
00221     int n = RECHERCHE_TRIPLE;
00222     int start = 20;
00223     bool trouve = false;
00224
00225     solution.clear();
00226     for(int flechettes=nbFlechettes; flechettes>0 && score>0;)
00227     {
00228         if(n == RECHERCHE_TRIPLE)
00229         {
00230             trouve = rechercherTriple(score, combinaison, start);
00231             if(!trouve)
00232             {
00233                 n = RECHERCHE_DOUBLE;
00234             }
00235         }
00236         else if(n == RECHERCHE_DOUBLE)
00237         {
00238             trouve = rechercherDouble(score, combinaison);
00239             if(!trouve)
00240             {
00241                 n = RECHERCHE_SIMPLE;
00242             }
00243         }
00244         else if(n == RECHERCHE_SIMPLE)
00245         {
00246             trouve = rechercherSimple(score, combinaison);
00247         }
00248
00249         if(trouve)
00250         {
00251             solution +=  " " + combinaison;
00252             flechettes--;
00253             if(score == 0)
00254             {
00255                 return true;
00256             }
```

```
00257         }
00258
00259         // pas trouvé ?
00260         if(flechettes == 0 && score != 0)
00261         {
00262             if(still)
00263                 return true;
00264             // on recommence
00265             solution.clear();
00266             score = s;
00267             flechettes = nbFlechettes;
00268             n++; // on change de tactique
00269         }
00270
00271         if(n == RECHERCHE_FINIE)
00272         {
00273             start--; // on essaye un autre triple
00274             // on recommence
00275             solution.clear();
00276             n = RECHERCHE_TRIPLE;
00277             score = s;
00278             flechettes = nbFlechettes;
00279         }
00280
00281         if(start == 1)
00282         {
00283             return false;
00284         }
00285     }
00286     return false;
00287 }
00288
00296 void Solution::trouverSolution(int scoreJoueur, int flechettes)
00297 {
00298     int score = 0;
00299     bool trouve = false;
00300     int nbFlechettes = 0;
00301     solution = "";
00302
00303     trouve = false;
00304     for(int i=BULL; i>0 && !trouve; i--)
00305     {
00306         // cibles inexistantes ?
00307         if(i > 20)
00308             continue;
00309         score = scoreJoueur; // <- le score à determiner
00310         nbFlechettes = flechettes; // <- le nombre de fléchettes
00311         if(extraireDouble(score, i))
00312         {
00313             nbFlechettes--;
00314             if(rechercher(score, nbFlechettes) || score == 0)
00315             {
00316                 solution += " D" + QString::number(i) + "*";
00317                 transmettreSolution(scoreJoueur);
00318                 trouve = true;
00319                 break;
00320             }
00321         }
00322     }
00323     if(!trouve)
00324     {
00325         rechercher(score, nbFlechettes+1, true);
00326         //transmettreSolution(scoreJoueur);       //activer pour avoir l'aide tout le temps même quand on
     ne peut pas finir
00327     }
00328 }
```

8.25 Référence du fichier solution.h

Déclaration de la classe solution (Module Ecran-DARTS)

```
#include <QObject>
#include <QString>
#include <QDebug>
```

Classes

— class Solution
Déclaration de la classe Solution (Module Ecran-DARTS)

Macros

- #define BULL 25
- #define RECHERCHE_DOUBLE 2
- #define RECHERCHE_FINIE 4
- #define RECHERCHE_SIMPLE 3
- #define RECHERCHE_TRIPLE 1

8.25.1 Description détaillée

Déclaration de la classe solution (Module Ecran-DARTS)

Version

0.3

Auteur

Bounoir Fabien

Définition dans le fichier solution.h.

8.25.2 Documentation des macros

8.25.2.1 BULL

```
#define BULL 25
```

Définition à la ligne 17 du fichier solution.h.

8.25.2.2 RECHERCHE_DOUBLE

```
#define RECHERCHE_DOUBLE 2
```

Définition à la ligne 21 du fichier solution.h.

Référencé par Solution : :rechercher().

8.25.2.3 RECHERCHE_FINIE

```
#define RECHERCHE_FINIE 4
```

Définition à la ligne 23 du fichier solution.h.

Référencé par Solution : :rechercher().

8.25.2.4 RECHERCHE_SIMPLE

```
#define RECHERCHE_SIMPLE 3
```

Définition à la ligne 22 du fichier solution.h.

Référencé par Solution : :rechercher().

8.25.2.5 RECHERCHE_TRIPLE

```
#define RECHERCHE_TRIPLE 1
```

Définition à la ligne 20 du fichier solution.h.

Référencé par Solution : :rechercher().

8.26 solution.h

```
00001 #ifndef SOLUTION_H
00002 #define SOLUTION_H
00003
00013 #include <QObject>
00014 #include <QString>
00015 #include <QDebug>
00016
00017 #define BULL                25
00018
00019 // Rechercher en premier ?
00020 #define RECHERCHE_TRIPLE    1
00021 #define RECHERCHE_DOUBLE    2
00022 #define RECHERCHE_SIMPLE    3
00023 #define RECHERCHE_FINIE     4
00024
00030 class Solution : public QObject
00031 {
00032     Q_OBJECT
00033 public:
00034     explicit Solution(QObject *parent = nullptr);
00035     ~Solution();
00036     void trouverSolution(int s, int flechettes);
00037
00038 private:
00039     QString solution;
00040
00041     void transmettreSolution(int score);
00042     bool aTriple(int points, const int score);
00043     bool rechercherTriple(int &score, QString &combinaison, int start);
00044     bool aDouble(int points, const int score);
00045     bool rechercherDouble(int &score, QString &combinaison);
00046     bool aSimple(int points, const int score);
00047     bool rechercherSimple(int &score, QString &combinaison);
00048     bool estDouble(int points, const int score);
00049     bool extraireDouble(int &score, int cible);
00050     bool rechercher(int score, int nbFlechettes, bool still=false);
00051
00052 signals:
00053     void solutionTrouver(QString solution);
00054
00055 public slots:
00056 };
00057
00058 #endif // SOLUTION_H
```