Rapport du projet informatique

Matthieu Denoux - Groupe 1

26 mars 2014

Table des matières

Première partie

Analyse du travail effectué

Chapitre 1

Présentation du sujet

1.1 Principe général

Le sujet choisi a pour intitulé **Babyfoot en réseau**. Il s'agit de concevoir un système complet de jeu en réseau. Le système serait donc séparé en deux parties, un serveur et un client. Chaque joueur pourrait donc se connecter à une partie n'ayant pas encore commencé et une fois le nombre de joueurs réunis (*2, 3 ou 4*), la partie serait lancée. Il faut donc réaliser à la fois le système réseau, l'interface graphique et imaginer un gameplay qui rende le jeu agréable.

1.2 L'interface graphique

Au niveau de l'interface graphique, il faudra réaliser plusieurs fenêtres successives de menus pour parvenir jusqu'au jeu lui-même. Ces fenêtres seront réalisées avec la bibliothèque inclue dans le package standard **Swing**. J'ai aussi choisi d'utiliser des JFrame et donc de programmer une application à part entière et non une appliquette.

Menu principal Un premier menu, dit *menu principal*, permettra de commencer une nouvelle partie en ligne, de rejoindre une partie déjà en cours, de modifier les quelques options disponibles ou bien de quitter le jeu. Un header unifie les différentes fenêtres, il donnera en plus une certaine identité graphique au jeu avec l'image affichée.

FIGURE 1.1 – Header du programme, très peu de recherche esthétique, simplement une image de base.

Commencer une partie Si l'on commence une nouvelle partie en ligne, on se retrouve dans une « salle de configuration ». Là, il est possible de configurer la partie que l'on souhaite lancer, l'ouvrir à d'autres joueurs, ce qui nous fait arriver dans une « salle d'attente »puis d'attendre que d'autres joueurs rejoignent la partie. Une fois que les équipes sont complètes, on peut lancer le jeu.

Rejoindre une partie On peut aussi sélectionner une partie dans la liste des parties déjà commencées, dans la limite des places disponibles. On rejoint alors une « salle d'attente »semblable à celle décrit dans la section précédente où l'on attend que la partie soit complète avant que le meneur, celui qui a créé la partie, ne lance le jeu. C'est le seul qui dispose de cette possibilité.

Déroulement d'une partie Le gameplay est entièrement manuel et voit le joueur maître de ses possibilités. Ainsi, les touches « A », « Z », « E », « R »permettent de sélectionner la canne que l'on souhaiterait manier. Le déplacement de la souris vers l'avant et l'arrière permettent quant à elle de déplacer les cannes tandis que les deux clics commandent le tir. Je n'ai pas eu le temps réaliser plusieurs puissances de tir possibles mais cela pourrait consister en une amélioration de la richesse du gameplay. Il est toutefois possible de retenir son tir.

Options Un petit menu sera consacré aux options, notamment la sensibilité du déplacement des barres que l'on souhaite utiliser. Un petit démonstrateur a été fait à côté afin de tester cette sensibilité.

Chapitre 2

Analyse de la solution envisagée

2.1 Découpage en modules

Je prévois de découper le code en deux grandes parties :
- Partie client
- Partie serveur

Les deux parties (client/serveur) seront elles-mêmes séparées en plusieurs modules.
- L'interface graphique (rangée dans /gui) qui sera absente du serveur
- La partie réseau de l'application (rangée dans /network), responsable de l'émission et de la réception des requêtes.
- Le cœur algorithmique de l'application (rangée dans /core) contenant les classes gérant les matchs, les joueurs, etc.

Chaque partie et chaque module comporteront donc plusieurs classes qui se chargeront chacune d'une des tâches du module parent.

2.2 L'interface graphique

MainFrame.java Gère la fenêtre qui englobe tout le reste. On utilisera en fait des JPanel pour modifier le contenu de cette fenêtre. J'empêcherai dans un premier temps de modifier la taille de la fenêtre pour éviter d'avoir des problèmes de dessin du terrain de babyfoot à gérer. Cette classe contient le **main** du programme. Elle permet d'accéder aux différents éléments : le client récepteur, les informations sur le joueur lui-même, etc.

BPanel.java Classe abstraite présentant certaines actions mécaniques quant aux caractéristiques des JPanel utilisés (header, background color, taille, layoutmanager, etc.). Elle contiendra notamment une référence vers l'instance de la fenêtre de type MainFrame La plupart des panels ci-dessous en héritent et peuvent donc accéder à cette classe.

ConnexionPanel.java Gère le premier écran affiché à l'ouverture du jeu. On y choisit le pseudonyme que l'on souhaite utiliser dans le reste du jeu et qui s'affichera ensuite sur les écrans lors du contact avec d'autres joueurs. Un message d'erreur est affiché en rouge lorsque le pseudo est déjà utilisé.

MenuPanel.java Gère le menu principal affiché après connexion. Contient les boutons qui mèneront vers les principales actions citées plus haut (nouvelle partie, rejoindre, options, quitter).

NewPanel.java Lorsque l'on lance une nouvelle partie, on obtient cet écran qui contiendra les principales options nécessaires pour configurer une partie puis l'ouvrir à des joueurs extérieurs. Une fois cette partie ouverte, on aboutit à une *salle d'attente* gérée par WaitingRoomPanel.

ServersPanel.java Gère la liste des parties actuellement en recherche de joueurs lorsque l'on cherche à rejoindre une partie. On sélectionne une partie dans cette liste puis on aboutit à une *salle d'attente*. La liste ne se rafraichit pas automatiquement mais un bouton est disponible en bas de page pour le faire.

WaitingRoomPanel.java Une fois la partie configurée et ouverte, on aboutit dans cette salle d'attente qui attendra que certaines conditions soient réunies pour permettre au jeu d'être commencé. On peut *aussi* y accéder depuis la partie *rejoindre une partie*. Il est alors possible d'utiliser le chat pour discuter avec les autres joueurs.

GamePanel.java La partie à proprement parler. Donc le conteneur du dessin du terrain qui gère les éléments extérieurs, les événements et toute autre interaction avec le reste du code. Inclut la zone de dessin décrite ci-dessous, le panel de chat dans la partie de droite et le panel d'informations sur le match dans celle de gauche.

GameZone.java Zone de dessin chargée de représenter le terrain, les joueurs, etc. Fichier assez conséquent puisqu'il contient toute la gestion de l'affichage et des données liées aux actions, déplacements pouvant être effectués par les joueurs sur le jeu. Toutes les informations, les calculs sont effectués par le serveur et envoyées au client qui, via cette classe, se charge de les concrétiser en affichant le jeu. L'affichage est géré par un Thread bien évidemment qui s'occupe de refaire une demande d'actualisation au serveur et affiche les données qu'il possède après la demande, quitte à être en retard d'un tour d'horloge. Ce fichier contient aussi la classe InfoZone qui affiche les informations (score, état) sur le match et donne la possibilité de mettre le jeu en pause et de le supprimer.

ChatPanel.java Une partie un peu à part qui est chargée de gérer l'affichage du chat. Celui-ci sera présent à de nombreux endroits : la liste des parties disponibles, la salle d'attente, la création d'une partie et durant le jeu lui-même. Or, tous ces affichages sont centralisés dans cet unique fichier qui centralise ainsi le traitement et l'affichage et permet de réutiliser à plusieurs endroits ces données.

SettingsPanel.java Cette classe assez light donne la possibilité de configurer les options du jeu. Je n'ai pas pris le temps d'en configurer beaucoup et il n'est pour l'instant disponible que la modification de sensibilité du jeu. Un démonstrateur a d'ailleurs été inclus à côté des boutons de modification pour tester la sensibilité. Deux boutons sont ensuite possibles : un qui sauvegarde la modification de sensibilité et un autre qui la sauvegarde et revient au menu principal simultanément.

2.3 Le réseau

2.3.1 Le principe général

Des messages sont échangés entre le client et le serveur et ce de façon répétée. Il faut donc utiliser des Thread pour gérer l'émission et la réception de données depuis le serveur et un

Thread pour gérer la réception côté client. Le serveur se chargera ensuite d'enregistrer les données statiques devant être conservées.

2.3.2 La partie client

ClientBabyfoot.java Il s'agit du main du client qui initialise les fenêtres et les différentes classes utilisées.

```
private static Player player ;
private static Chat chat ;
private static Client client ;
```

Client.java Cette classe gère la connexion au serveur : il y a en tout trois connexions (une pour le tchat, une pour les joueurs et une pour les données des matchs) afin de pouvoir envoyer plusieurs requêtes simultanément du même client. Toutes les requêtes envoyées sont sous la forme de chaîne de caractères avec l'utilisation d'un caractère spécial, qui est contenu dans une constante de la classe Utils du Core comme séparateur.

```
private static Socket socketChat ;
private static Socket socketPlayer ;
private static Socket socketMatch ;
private static Socket socketGame ;
private ChatClient cc ;
private PlayerClient pc ;
private MatchClient mc ;
private GameClient gc ;
private Thread tChat ;
private Thread tPlayer ;
private Thread tMatch ;
private Thread tGame ;
```

PlayerClient.java Gère les différentes actions possibles par et sur les joueurs et les requêtes envoyées au serveur pour pouvoir satisfaire le système (connexion d'un joueur, déconnecter un joueur, ajouter un match, etc.). Elle contient aussi un Thread qui écoute l'entrée de la socket.

MatchClient.java Procède de la même façon que PlayerClient pour les matchs, envoie les requêtes au serveur et récupère les données sur la position de la balle, des barres des différents joueurs, etc.

ChatClient.java S'occupe de gérer la partie client du tchat : une fois instanciée par le ChatPanel, elle permet d'envoyer des données au serveur et de récupérer la liste des messages disponibles dans le salon. Elle permet aussi de changer de salon, d'afficher la liste des salons et celle des joueurs connectés sur le jeu.

2.3.3 La partie serveur

AbstractServer.java Classe abstraite qui gère quelques méthodes de base que doivent toutes présenter les classes gérant les serveurs.

ServerBabyfoot.java Gère les différentes requêtes et les redirige vers les autres entités du serveur (chat, jeu, etc.). Contient un système de Thread pour pouvoir gérer simultanément ces différentes actions, un Thread général qui gère les requêtes. C'est aussi le main qui lance le serveur.

```
ServerSocket socketserver = null;
Socket socket = null;
BufferedReader in;
PrintWriter out;
String login;
public static ChatServer tchat;
public static PlayerServer tplayer;
public static MatchServer tmatch;
```

PlayerServer.java Gère la connexion au serveur d'un des joueurs. L'enregistre dans une liste de joueurs connectés. Peut aussi renvoyer la liste des joueurs connectés actuellement et leur état.

MatchServer.java Gère tous les échanges de données sur les matchs. S'occupe ensuite de fournir les informations nécessaires au client sur l'état actuel de la position de la balle par exemple. Nécessitera peut-être, pour des questions de réactivité, d'être multi-threadé.

ChatServer.java S'occupe de gérer la partie serveur du tchat : écoute un port, récupère les messages envoyés par des joueurs ainsi que le salon où ils se trouvaient à ce moment là. Les enregistre dans un fichier ou bien dans une base de données (à voir).

2.4 Le cœur algorithmique

Player.java Le bloc de base du cœur algorithmique est le joueur. Il est instancié par le serveur et a plusieurs attributs : son login, son état actuel (s'il est en train de jouer ou non). Si le joueur est en match, la classe contient aussi les barres qu'il a le droit de déplacer.

```
private Match match;
private String login;
```

Utils.java Est une classe abstraite qui gère les fonctions utilitaires comme la mise-en-forme de la date, la mise-en-forme des requêtes, les fonctions de hash utilisées pour vérifier la validité des requêtes, etc. Commune au client et au serveur. Cette classe contient surtout les constantes du programme ainsi que les types créés (enum) pour repérer le côté du joueur, les barres, le statut du match, le type de collision, etc.

```
public static enum Types  ONEVSONE, TWOVSTWO, ONEVSTWO;
public static enum States  WAITING, FULL, PLAYING, FINISHED;
public static enum Sides  DOWN, UP;
public static final int MATCH_END = 100;
public static enum Rod  GARDIEN , DEFENSE, MILIEU, ATTAQUE;
public static enum RodStatus  NORMAL, SHOOTING, HOLDING;
public static enum CollisionType  SIDES, UPANDDOWN;
public static final String SEPARATOR = ";";
```

public static final int GOAL_SIZE = 2*100 ;
public static final int LINE_STRENGTH = 4 ;
public static final int GAP_EDGE = 2*20 ;
public static final int IMAGE_PLAYER_Y = 38 ;
public static final int IMAGE_PLAYER_X = 30 ;
public static final int MOVE_STEP = 10 ;
public static final int BALL_RADIUS = 15 ;
public static final int HEIGHT = 700 ;
public static final int WIDTH = 900 ;
public static final int MAX_INITIAL_SPEED = 4 ;
public static final int GARDIEN_POSITION = GAP_EDGE+30 ;
public static final int DEFENSE_POSITION = GAP_EDGE+30+100 ;
public static final int MILIEU_POSITION = (WIDTH-LINE_STRENGTH)/2-70 ;
public static final int ATTAQUE_POSITION = WIDTH-Utils.LINE_STRENGTH-Utils.GAP_EDGE-230 ;

Match.java Contient les données principales pour une partie, notamment l'avancement de la partie, l'état des scores, les joueurs y participant, etc. Sera appelée par le serveur. Cette classe contient aussi toutes les informations sur les données factuelles d'une partie, à savoir la position de la balle, la position des barres, etc. Elle se charge d'effectuer l'ensemble des calculs de déplacements de la balle, comptabilise les buts, met en place les pauses.

private int leftScore ;
private int rightScore ;
private Types type ;
private States state ;
private Player player1 ;
private Player player2 ;
private Player player3 ;
private Player player4 ;
private float ballX ;
private float ballY ;
private float ballSpeedX ;
private float ballSpeedY ;
private Collisions collisions ;
private boolean noSlow = true ;
private boolean pause = false ;
private final int STEP_X = 2 ;
private final int STEP_Y = 2 ;
private int status = 0 ;

Collisions.java Gère tout ce qui touche aux collisions, avec les joueurs ou les bords. C'est là que se situe toute la difficulté de calcul. Elle contient donc des attributs pour la gestion de la balle ainsi qu'une Hashtable à deux niveaux stockant les dernières collisions. L'intérêt de la HashTable est ici clairement qu'elle permet de stocker les temps en fonction de la barre et du côté où est situé le joueur.

```
private float ballX;
private float ballY;
private float ballSpeedX;
private float ballSpeedY;
private Hashtable<Sides, Hashtable<Rod, Long>>lastCollision;
```

Database.java Aucune base de données n'a finalement été utilisée. Le serveur stocke donc ses données dans la RAM puis les supprime lors de sa fermeture. Dans un souci de pérennité et de performances, il serait intéressant de les stocker dans des fichiers textes que l'on puisse ainsi conserver des archives.

Chapitre 3

Échéancier

Ce qui était prévu

1. **Fin Décembre** : rédiger l'avant-projet.
2. **Début/Mi Janvier** : obtenir la validation et les annotations sur la structure choisie.
3. **Fin Janvier** : réaliser la partie graphique du programme et avoir regardé les grandes lignes du développement serveur / bases de données. Avoir mis au point les éléments de base du gameplay (interaction joueur/machine).
4. **Fin Février** : développer le serveur et la gestion des différents types de requêtes. Mettre au point le tchat et la gestion des joueurs avec la base de données.
5. **Mi Mars** : dresser les liens entre serveur et jeu. Tester.

Ce qui a été fait

1. **Fin Décembre** : rédiger l'avant-projet.
2. **Fin décembre/Début Janvier** : développer la partie réseau et l'architecture de base des requêtes.
3. **Fin Janvier** : réalisation de l'interface graphique et des premiers calculs pour la mise-en-place de matchs.
4. **Février** : ajout des collisions et corrections des principaux bugs liés à la gestion des joueurs, matchs et chat.
5. **Mars** : débuguer le reste du programme et ajouter les fonctionnalités supplémentaires nécessaires à un bon fonctionnement.

Deuxième partie

Explications des choix techniques

3.1 Le réseau

Le réseau est au coeur de mon projet et j'ai donc développé plusieurs Thread en parallèle pour pouvoir gérer plus facilement l'envoi de messages par mon application. Il a fallu mettre au point une nomenclature des messages afin d'obtenir successivement le type de message envoyé et le domaine qui est atteint, donc la classe effectuant le traitement de la requête, puis le joueur concerné (c'est le login qui est utilisé ici bien qu'un système d'ID aurait pu être mis en place) et enfin les données propres à la requête.

De plus, le serveur recevant un grand nombre de requêtes, j'ai séparé le serveur en plusieurs entités afin de faciliter le traitement. On trouve ainsi une classe dédiée entièrement à la gestion du Chat et des requêtes échangées. Il y a aussi un serveur se chargeant de gérer les joueurs et enfin, le plus gros des trois, un serveur s'occupant des requêtes de jeux. Il est assez sollicité puisqu'une requête est envoyée aux 2, 3, 4 clients toutes les 20 ms environ (un peu plus parfois). Ces requêtes peuvent être assez longues puisqu'elles contiennent de nombreuses informations : la position des barres, leur statut (normal, levée, tirant) ce qui représente 16 coordonnées puis les huit barres.

3.2 La gestion des barres

Un point a été intéressant du point de vue algorithmique, c'est la gestion des barres. Suivant le type de match, un joueur a accès à certaines barres. Par exemple, quand il est seul contre deux joueurs, il a accès à toutes les barres. Par contre, s'il n'est pas seul, le calcul est fait automatiquement selon l'ordre d'arrivée des joueurs dans la partie. Ce qui a été intéressant, c'est de trouver une façon de stocker ces informations. Il fallait ici obtenir une méthode simple pour définir « l'utilisation »des barres, lesquelles sont occupées et lesquelles sont libres. Ce problème est très similaire à un problème d'autorisations et de droits. En effet, a-t-on le droit ou non de bouger les barres ?

Pour cela, ma technique a été très simple, c'est d'utiliser la notation pour binaire et associer à chaque bit une autorisation particulière. Je code ici les autorisations sur quatre bits, chacun représentant deux barres, d'un côté ou d'un autre. Ainsi, les deux premiers bits (valeurs correspondant à 2^0 et 2^1) correspondent aux barres du bas et les deux autres à celles du haut. Lorsque le statut des barres est actualisé, *i.e.* qu'un joueur entre dans la partie, cet entier est testé pour situer dans quel cas se situe le jeu et où des barres sont disponibles puis l'entier d'utilisation est mis à jour en ajoutant 1, 2, 3, 4, etc. pour signifier quelles barres va prendre le joueur. De même lorsqu'un joueur quitte la partie, l'entier voit sa valeur soustraite par celle représentant le joueur. On a ainsi une façon simple d'accéder à la possibilité pour un joueur de toucher à une barre puisqu'il s'agit d'utiliser les opérateurs binaires pour tester une autorisation. Ce nombre est stocké par le serveur et est transmis très facilement aux clients qui stockent eux les autorisations dans une Hashtable.

Il eut été possible de s'affranchir de toutes ces considérations en réalisant un écran supplémentaire avant le début de la partie où le joueur hôte de la partie aurait tout simplement assigné telle ou telle barre à tel ou tel joueur. La gestion se serait alors faite avec une Hashtable contenant des noms de joueurs par exemple. Il aurait été toutefois intéressant de stocker la même information de « l'utilisation »des barres afin de vérifier que toutes les barres ont bien été assignées à quelqu'un.

3.3 La gestion du jeu

Pour calculer la position de la balle au cours du temps, j'ai utilisé un Thread qui s'actualise toutes les 10ms et qui effectue à chaque tour d'horloge l'actualisation de la position à partir de la vitesse et des tests de collisions. Les tests de collisions sont calculés à part dans une classe appelée Collision. Je vérifie ainsi la position de la balle par rapport aux bords et détermine si une collision a eu lieu avec une de ces limites. Une fonction est appelée aussi afin de déterminer si il n'y a pas eu de contact avec un des joueurs du terrain.

La principale difficulté algorithmique vient de cet aspect justement. Parvenir à prévoir les différents comportements de la balle par rapport aux joueurs. Les coordonnées utilisées par ces calculs diffèrent en plus parfois de celles d'affichage et il faut prendre en compte la taille de l'écran ainsi que les imprécisions dues au caractère discret du calcul. J'ai encore parfois des problèmes de collisions qui se produisent trop tôt, c'est-à-dire que le joueur humain a l'impression qu'elle se cogne contre un objet invisible, ou encore des collisions gérées trop tard auquel cas la balle rentre partiellement dans le joueur en plastique.

Une autre difficulté a été de déterminer dans quel sens va la balle après le rebond. En effet, il faut parvenir à placer la balle dans une des quatre zones déterminées par les bissectrices issues des coins du joueur. Si la balle arrive sur les faces avant ou arrière du joueur, elle doit donc voir sa vitesse horizontale changer de sens. Si elle arrive sur les faces de côté, elle doit alors voir sa vitesse verticale changer. Il faut donc calculer la pente de la droite formée par les deux points suivants, le centre de la balle et un des quatre coins du joueur que l'on cherche à étudier, puis la comparer avec la pente de la droite qui marque la séparation. Cela permet ainsi de procéder au rebond.

Le problème du caractère discret de ces calculs, c'est que les corrections ne suffisent pas à faire sortir la balle de la zone de collision. Ainsi, malgré le changement de vitesse et la correction effectuée lorsque la balle entre dans une des zones où la collision a lieu, il est possible que la balle reste dans une de ces zones auquel cas, le prochain tour de calcul risque de demander un nouveau changement de vitesse ce qui est bien sûr absurde.

Un des derniers défis techniques a été de régler la sensibilité des barres afin de permettre aux joueurs d'avoir à la fois un jeu réactif, confortable, c'est-à-dire où la souris n'a pas besoin de beaucoup bouger, et en même temps quelque chose de suffisamment précis pour pouvoir shooter dans la balle au bon endroit au bon moment.

J'ai essayé de mettre au point un amortissement de la vitesse de la balle au cours du jeu. Il est difficile de s'en occuper car, en conservant quelque chose de significatif, il arrive que la balle s'immobilise ce qui est légérement problématique.

Troisième partie

Le code partiel du programme

Chapitre 1

Client

1.1 Core

1.1.1 ClientBabyfoot.java

```
1  /** Cette classe contient le Main du client du projet. Elle initialise les
       différents composants nécessaires à l'exécution
2   * du programme, c'est-à-dire le client, pour la connexion avec le serveur,
       le système de chat et le joueur. Elle lance
3   * ensuite la classe MainFrame du package gui.
4   */
5  public class ClientBabyfoot {
6          private Player player;
7          private Chat chat;
8          private Client client;
9          private MainFrame mainFrame;
10
11         private int port;
12
13         public JPanel actualPanel;
14
15         public ClientBabyfoot(int port) {
16                 this.port = port;
17         }
18
19         public static void main(String[] args){
20                 int port = 2010;
21                 if( args.length >= 1 ){
22                         if( args[0].equals("-port") && args[1].length() > 0
                               ){
23                                 port = Integer.valueOf(args[1]);
24                         }
25                 }
26                 ClientBabyfoot m = new ClientBabyfoot(port);
27                 m.init();
28         }
29
30         /** Lors de la fermeture de la fenêtre, supprime le joueur du
               serveur puisqu'il s'est déconnecté. */
31         public void closeWindow() {
32                 if( !getPlayer().getLogin().equals("") ){
33                         try{
34                                 getPlayer().removePlayer(getPlayer().
                                       getLogin());
35                         }catch(Exception e){
```

```
                                e.printStackTrace();
                        }
                }
                System.exit(0);
        }

        public void init(){
                setClient(new Client(this));
                setChat(new Chat(this));
                setPlayer(new Player(this));
                mainFrame = new MainFrame("Babyfoot en réseau trololol",
                    this);
        }
```

1.1.2 Utils.java

```
/** Cette classe dipose de fonctions utiles qui peuvent être nécessaires par
    toutes les parties du programme. Elle contient aussi
les constantes nécessaires (par exemple de type enum) concernant le
   programme. */
public abstract class Utils {
        public static enum Types { ONEVSONE, TWOVSTWO, ONEVSTWO };
        public static enum States { WAITING, FULL, PLAYING, FINISHED };
        public static enum Sides { DOWN, UP };

        public static final int MATCH_END = 100;

        public static enum Rod { GARDIEN , DEFENSE, MILIEU, ATTAQUE };
        public static enum RodStatus { NORMAL, SHOOTING, HOLDING };

        public static enum CollisionType { SIDES, UPANDDOWN };

        public static final String SEPARATOR = ";";

        public static final int GOAL_SIZE = 2*100;
        public static final int LINE_STRENGTH = 4;
        public static final int GAP_EDGE = 2*20;
        public static final int IMAGE_PLAYER_Y = 38;
        public static final int IMAGE_PLAYER_X = 30;
        public static final int MOVE_STEP = 10;
        public static final int BALL_RADIUS = 15;

        public static final int HEIGHT = 700;
        public static final int WIDTH = 900;

        public static final int MAX_INITIAL_SPEED = 4;

        public static final int GARDIEN_POSITION = GAP_EDGE+30;
        public static final int DEFENSE_POSITION = GAP_EDGE+30+100;
        public static final int MILIEU_POSITION = (WIDTH-LINE_STRENGTH)
            /2-70;
        public static final int ATTAQUE_POSITION = WIDTH-Utils.LINE_STRENGTH
            -Utils.GAP_EDGE-230;


        private static int sensibility = 30;


        @SuppressWarnings("serial")
```

```
40          public static final Hashtable<Rod, Integer> Y_STAGGERING_DEFAULT =
                new Hashtable<Rod, Integer>(){{ put(Rod.GARDIEN, 100); put(Rod.
                DEFENSE, 150 ); put(Rod.MILIEU, 100); put(Rod.ATTAQUE, 100); } };
41
42          public static final int IMAGE_PLAYER_SHOOTING_X = 30;
43
44
45          public static int getYPositionPlayer( Hashtable<Rod, Integer>[]
                yDecal, Rod rod, int i, int nb, Sides side ){
46              return yDecal[side == Sides.UP ? 1 : 0].get(rod)-Utils.
                    Y_STAGGERING_DEFAULT.get(rod)+i*Utils.HEIGHT/(1+nb)-Utils
                    .IMAGE_PLAYER_Y/2;
47          }
48
49          public static int getYPositionPlayer( int position, Rod rod, int i,
                int nb ){
50              return position-Utils.Y_STAGGERING_DEFAULT.get(rod)+i*Utils.
                    HEIGHT/(1+nb)-Utils.IMAGE_PLAYER_Y/2;
51          }
52
53
54          /** Hache un mot de passe en se basant sur le principe de l'
                algorithme MD5. Retourne la chaine.
55          @param s contient le mot de passe à chiffrer. */
56          public static String hash(String s){
57
58              byte[] bytes = s.getBytes();
59              byte[] hashTable = null;
60              try {
61                      hashTable = MessageDigest.getInstance("MD5").digest(
                            bytes);
62              }
63              catch(NoSuchAlgorithmException e){
64
65              }
66              StringBuffer hashString = new StringBuffer();
67              int n = hashTable.length;
68              for (int i = 0; i < n; i++)
69              {
70              String hex = Integer.toHexString(hashTable[i]);
71           if (hex.length() == 1){
72              hashString.append('0');
73              hashString.append(hex.charAt(hex.length() - 1));
74           }
75           else
76              hashString.append(hex.substring(hex.length() - 2));
77              }
78              return new String(hashString);
79          }
80
81          public static void main( String[] args ){
82              System.out.println(Utils.hash("
                    bobabcdefghijklmnopqrstuvwxyz1234567890.!:;,"));
83          }
84
85          /** Formate un array en renvoyant un array. C'est nécessaire lorsque
                le paramètre contient des chaines
86          avec des informations concaténées par des tirets. L'array de sortie
                ne contient que ces informations.
```

```
87          @param list est un ensemble de chaines de caractères de la forme "
               foo - bar". Le programme récupère bar et renvoie le tableau des "
               bar" */
88          public static String[] formatStringArray(String[] list){
89                  for(int i = 0; i<list.length; i++){
90                          if( !list[i].equals("Pas de messages") ){
91                                  String[] m = list[i].split(" - ",2);
92                                  list[i] = m[1];
93                          }
94                  }
95                  return list;
96          }
97
98          public static String getChatServerNameFromHost(String loginHost){
99                  return "Partie de " + loginHost;
100         }
101
102
103         /** Retourne une date formatée à partir d'une chaine de caractère de
                type timestamp.
104         @param date est de type Timestamp */
105         public static String formatDate(String date){
106                 return formatDate(Integer.valueOf(date));
107         }
108
109         /** Retourne une date formatée à partir d'une chaine de caractère de
                type timestamp.
110         @param date est de type Timestamp */
111         public static String formatDate(int date){
112                 Date d = new Date((long)date);
113                 SimpleDateFormat formatter = new SimpleDateFormat("dd/MM hh:
                       mm:ss");
114                 return formatter.format(d);
115         }
116
117         /** Affiche un array */
118         public static void printArray(String[] s){
119                 for( int i = 0; i<s.length; i++){
120                         System.out.println(i + ". " + s[i] );
121                 }
122         }
123
124         /** Affiche un array de à deux dimensions */
125         public static void printArray(int[][] s){
126                 for( int i = 0; i<s.length; i++){
127                         for( int j = 0; j<s[0].length; j++){
128                                 System.out.println(i + ". " + s[i][j] );
129                         }
130                 }
131         }
```

1.2 Gui

1.2.1 BPanel.java

```
1   /** Cette classe abstraite permet simplement de garder quelques données
       communes à tous les panels qui seront utilisés dans l'IHM
```

```
 par la suite : le LayoutManager qui est un BorderLayout, la couleur du fond
     (blanche) et la présence du header (cf. @Header) dans la zone NORTH
 du BorderLayout. */
public abstract class BPanel extends JPanel {
        private MainFrame window;

        public BPanel(MainFrame f, boolean display, int heightHeader ){
                setWindow(f);
                getWindow().getMain().actualPanel = this;
                setLayout(new BorderLayout());
                setBackground(Color.WHITE);
                add(new Header(display, heightHeader),BorderLayout.NORTH);
        }

        public BPanel(MainFrame f ){
                this( f, true, 150 );
        }
        public MainFrame getWindow() {
                return window;
        }
        public void setWindow(MainFrame window) {
                this.window = window;
        }
}
```

1.2.2 GameZone.java

```
public class GameZone extends JPanel implements KeyListener,
   MouseMotionListener, MouseListener {
        private static final long serialVersionUID = 1L;
        /**
         * Taille des buts en pixels et épaisseur des traits
         */
        private int oldY = 0;

        private Hashtable<Rod, Integer>[] yPosition;

        private Hashtable<Rod, Boolean> rodPositions;

        private Hashtable<Rod, RodStatus>[] rodStatus;

        Rod rodPosition;

        private int ballX;
        private int ballY;


        private GamePanel gamepanel;
        private InfoZone infoZone;

        private long shootBeginning;
        private boolean pause = false;
        private Sides side;
        private int lastKeyY;
        private Sides toNormalSide;
        private Rod toNormalRod;

        @SuppressWarnings("unchecked")
        public GameZone(GamePanel window, boolean testMode, int width){
```

```
32                  this.gamepanel = window;
33
34                  this.setSide(getGamePanel().getWindow().getPlayer().getSide
                        ());
35                  setSize(width,729);
36
37                  infoZone = new InfoZone(this);
38                  setRodStatus(new Hashtable[2]);
39                  rodStatus[0] = new Hashtable<Rod, RodStatus>();
40                  rodStatus[0].put(Rod.GARDIEN, RodStatus.NORMAL);
41                  rodStatus[0].put(Rod.DEFENSE, RodStatus.NORMAL);
42                  rodStatus[0].put(Rod.MILIEU, RodStatus.NORMAL);
43                  rodStatus[0].put(Rod.ATTAQUE, RodStatus.NORMAL);
44                  rodStatus[1] = new Hashtable<Rod, RodStatus>();
45                  rodStatus[1].put(Rod.GARDIEN, RodStatus.NORMAL);
46                  rodStatus[1].put(Rod.DEFENSE, RodStatus.NORMAL);
47                  rodStatus[1].put(Rod.MILIEU, RodStatus.NORMAL);
48                  rodStatus[1].put(Rod.ATTAQUE, RodStatus.NORMAL);
49
50                  yPosition = new Hashtable[2];
51                  yPosition[0] = new Hashtable<Rod, Integer>();
52                  yPosition[0].put(Rod.GARDIEN, Utils.Y_STAGGERING_DEFAULT.get
                        (Rod.GARDIEN));
53                  yPosition[0].put(Rod.DEFENSE, Utils.Y_STAGGERING_DEFAULT.get
                        (Rod.DEFENSE));
54                  yPosition[0].put(Rod.MILIEU, Utils.Y_STAGGERING_DEFAULT.get(
                        Rod.MILIEU));
55                  yPosition[0].put(Rod.ATTAQUE, Utils.Y_STAGGERING_DEFAULT.get
                        (Rod.ATTAQUE));
56                  yPosition[1] = new Hashtable<Rod, Integer>();
57                  yPosition[1].put(Rod.GARDIEN, Utils.Y_STAGGERING_DEFAULT.get
                        (Rod.GARDIEN));
58                  yPosition[1].put(Rod.DEFENSE, Utils.Y_STAGGERING_DEFAULT.get
                        (Rod.DEFENSE));
59                  yPosition[1].put(Rod.MILIEU, Utils.Y_STAGGERING_DEFAULT.get(
                        Rod.MILIEU));
60                  yPosition[1].put(Rod.ATTAQUE, Utils.Y_STAGGERING_DEFAULT.get
                        (Rod.ATTAQUE));
61
62                  rodPosition = ( getGamePanel().getWindow().getMain().
                        getPlayer().getRodAvailables().get(Rod.MILIEU) ? Rod.
                        MILIEU : Rod.GARDIEN );
63              setPreferredSize(new Dimension(width,729));
64              setMinimumSize(new Dimension(width,729));
65
66              addKeyListener(this);
67              addMouseListener(this);
68              addMouseMotionListener(this);
69
70              setFocusable(true);
71              requestFocusInWindow();
72
73              if( !testMode ){
74                      Thread tr = new Thread(new RefreshRods(this));
75                      tr.start();
76              }
77          }
78
79
80          public void paint(Graphics g){
```

```
81                  g.setColor(new Color(116,152,29));
82                  //g.fillRect(0,0,getWidth(),getHeight());
83                  try {
84                Image img = ImageIO.read(new File("pictures/terrain.png"));
85                g.drawImage(img, 0, 0, this.getWidth(), this.getHeight(), this
                      );
86                  } catch (IOException e) {
87                e.printStackTrace();
88              }
89
90                  initInfoZone();
91                  drawLines(g);
92                  drawBall(g);
93                  drawGoals(g);
94                  drawPlayers(g);
95                  drawRodPosition(g);
96          }
97
98          private void initInfoZone(){
99                  infoZone.getLeftScore().setBackground(Color.RED);
100                 infoZone.getRightScore().setBackground(Color.BLUE);
101                 gamepanel.repaint();
102         }
103
104         private void drawBall(Graphics g) {
105                 //Côté gauche
106                 g.setColor(Color.WHITE);
107                 g.fillOval(ballX-Utils.BALL_RADIUS, ballY-Utils.BALL_RADIUS,
                        Utils.BALL_RADIUS*2, Utils.BALL_RADIUS*2);
108         }
109
110
111         private void drawGoals(Graphics g){
112                 //Côté gauche
113                 g.setColor(Color.RED);
114                 //Long côté
115                 g.fillRect(Utils.GAP_EDGE/2,Utils.HEIGHT/2-Utils.GOAL_SIZE
                        /2,Utils.LINE_STRENGTH,Utils.GOAL_SIZE);
116                 //petits côtés
117                 g.fillRect(Utils.GAP_EDGE/2,Utils.HEIGHT/2-Utils.GOAL_SIZE
                        /2,Utils.GAP_EDGE/2,Utils.LINE_STRENGTH);
118                 g.fillRect(Utils.GAP_EDGE/2,Utils.HEIGHT/2+Utils.GOAL_SIZE
                        /2,Utils.GAP_EDGE/2,Utils.LINE_STRENGTH);
119                 //Côté droit
120                 g.setColor(Color.BLUE);
121                 //Long côté
122                 g.fillRect(Utils.WIDTH-Utils.GAP_EDGE/2-Utils.LINE_STRENGTH,
                        Utils.HEIGHT/2-Utils.GOAL_SIZE/2,Utils.LINE_STRENGTH,
                        Utils.GOAL_SIZE);
123                 //petits côtés
124                 g.fillRect(Utils.WIDTH-Utils.GAP_EDGE,Utils.HEIGHT/2-Utils.
                        GOAL_SIZE/2,Utils.GAP_EDGE/2,Utils.LINE_STRENGTH);
125                 g.fillRect(Utils.WIDTH-Utils.GAP_EDGE,Utils.HEIGHT/2+Utils.
                        GOAL_SIZE/2,Utils.GAP_EDGE/2,Utils.LINE_STRENGTH);
126         }
127
128         private void drawLines(Graphics g){
129                 try {
130               Image img = ImageIO.read(new File("pictures/bordhaut.png"));
131               g.drawImage(img, 0, 0, this);
```

```
132                 } catch (IOException e) {
133               e.printStackTrace();
134             }
135                 try {
136               Image img = ImageIO.read(new File("pictures/bordbas.png"));
137               g.drawImage(img, 0, Utils.HEIGHT - 46, this);
138                 } catch (IOException e) {
139               e.printStackTrace();
140             }
141
142                 g.setColor(Color.WHITE);
143                 //Milieu de terrain
144                 g.fillRect((Utils.WIDTH-Utils.LINE_STRENGTH)/2,46,Utils.
                        LINE_STRENGTH,Utils.HEIGHT-92);
145                 //gauche
146                 g.fillRect(Utils.GAP_EDGE,46,Utils.LINE_STRENGTH,Utils.
                        HEIGHT-92);
147                 //haut
148                 //g.fillRect(0,Utils.GAP_EDGE,Utils.WIDTH,Utils.
                        LINE_STRENGTH);
149                 //droite
150                 g.fillRect(Utils.WIDTH-Utils.GAP_EDGE-Utils.LINE_STRENGTH
                        ,46,Utils.LINE_STRENGTH,Utils.HEIGHT-92);
151                 //bas
152                 //g.fillRect(0,Utils.HEIGHT-Utils.GAP_EDGE-Utils.
                        LINE_STRENGTH,Utils.WIDTH,Utils.LINE_STRENGTH);
153         }
154
155         private void drawPlayers(Graphics g){
156                 drawPlayer(g, Utils.GARDIEN_POSITION, 0, 1, Color.RED, Sides
                        .DOWN, Rod.GARDIEN);
157                 drawPlayer(g, Utils.DEFENSE_POSITION, 0, 2, Color.RED, Sides
                        .DOWN, Rod.DEFENSE);
158                 drawPlayer(g, Utils.MILIEU_POSITION, 0, 5, Color.RED, Sides.
                        DOWN, Rod.MILIEU);
159                 drawPlayer(g, Utils.ATTAQUE_POSITION, 0, 3, Color.RED, Sides
                        .DOWN, Rod.ATTAQUE);
160
161                 drawPlayer(g, Utils.WIDTH-Utils.LINE_STRENGTH-Utils.
                        GARDIEN_POSITION, 0, 1, Color.RED, Sides.UP, Rod.GARDIEN)
                        ;
162                 drawPlayer(g, Utils.WIDTH-Utils.LINE_STRENGTH-Utils.
                        DEFENSE_POSITION, 0, 2, Color.RED, Sides.UP, Rod.DEFENSE)
                        ;
163                 drawPlayer(g, Utils.WIDTH-Utils.LINE_STRENGTH-Utils.
                        MILIEU_POSITION, 0, 5, Color.RED, Sides.UP, Rod.MILIEU);
164                 drawPlayer(g, Utils.WIDTH-Utils.LINE_STRENGTH-Utils.
                        ATTAQUE_POSITION, 0, 3, Color.RED, Sides.UP, Rod.ATTAQUE)
                        ;
165         }
166
167         /**
168          * side : true = orienté vers la gauche, false orienté vers la
                droite.
169          * position : quelle est la position de la barre ? tir, droit, etc.
170          */
171
172         private void drawPlayer(Graphics g, int x, int y, int nb, Color
               color, Sides side, Rod rod ){
```

```
                RodStatus status = rodStatus[side == Sides.UP ? 1 : 0].get(
                    rod);
                int position = status == RodStatus.HOLDING ? 3 : (status ==
                    RodStatus.NORMAL ? 1 : 2);
                g.setColor(new Color(192, 192, 192));
                g.fillRect(x,20,3*Utils.LINE_STRENGTH/2,Utils.HEIGHT-92+52);
                g.setColor(color);
                for( int i=1; i<=nb;i++){
                        try {
                                Image img = null;
                                if(side == Sides.UP)
                                        img = ImageIO.read(new File("
                                            pictures/joueurdroit" + String.
                                            valueOf(position) + ".png"));
                                else
                                        img = ImageIO.read(new File("
                                            pictures/joueurgauche" + String.
                                            valueOf(position) + ".png"));
                                if(position==1)
                                        g.drawImage(img, x-Utils.
                                            IMAGE_PLAYER_X/3, Utils.
                                            getYPositionPlayer(yPosition, rod
                                            , i, nb, side), this);
                                else if(position==2 && side == Sides.DOWN )
                                        g.drawImage(img, x-Utils.
                                            IMAGE_PLAYER_X/3-30, Utils.
                                            getYPositionPlayer(yPosition, rod
                                            , i, nb, side), this);
                                else if(position==2 && side == Sides.UP )
                                        g.drawImage(img, x-Utils.
                                            IMAGE_PLAYER_X/3-60, Utils.
                                            getYPositionPlayer(yPosition, rod
                                            , i, nb, side), this);
                                else if(position==3 && side == Sides.DOWN )
                                        g.drawImage(img, x-Utils.
                                            IMAGE_PLAYER_X/3-40, Utils.
                                            getYPositionPlayer(yPosition, rod
                                            , i, nb, side), this);
                                else if(position==3 && side == Sides.UP )
                                        g.drawImage(img, x-Utils.
                                            IMAGE_PLAYER_X/3-40, Utils.
                                            getYPositionPlayer(yPosition, rod
                                            , i, nb, side), this);
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                }
        }
```

```
1
2   class RefreshRods implements Runnable {
3           private GameZone gamezone;
4
5           public RefreshRods(GameZone g ){
6                   gamezone = g;
7           }
8
9           public void run() {
10                  int i = 0;
11                  while(true){
```

```
12                          //Alias pour faciliter la lecture
13                          GameClient gc = gamezone.getGamePanel().getWindow().
                                getMain().getClient().getGc();
14                          if( gamezone.getGamePanel().getWindow().getMain().
                                getClient().getMc().isToDelete() ){
15                                  gamezone.getGamePanel().getWindow().
                                        setContentPane(new MenuPanel(gamezone.
                                        getGamePanel().getWindow()));
16                                  gamezone.getGamePanel().getWindow().
                                        setVisible(true);
17                                  break;
18                          }
19                          Player p = gamezone.getGamePanel().getWindow().
                                getMain().getPlayer();
20                          gamezone.refreshPositions( gc.getPositions( p.
                                getLogin(), false ) , p.getSide(), gc.getBallX(),
                                 gc.getBallY() );
21                          MatchClient mc = gamezone.getGamePanel().getWindow()
                                .getMain().getClient().getMc();
22                          if( mc.getLeftScore() < 20 && mc.getLeftScore() >= 0
                                 )
23                                  gamezone.getLeftScore().setText( " " + mc.
                                        getLeftScore() );
24                          if( mc.getRightScore() < 20 && mc.getRightScore() >=
                                 0 )
25                                  gamezone.getRightScore().setText( " " + mc.
                                        getRightScore() );
26                          gamezone.setPause(mc.isPause());
27                          try{
28                                  Thread.sleep(10);
29                          }catch( InterruptedException e ){
30
31                          }
32                          if( gamezone.getToNormalSide() != null && gamezone.
                                getToNormalRod() != null ){
33                                  if( i > 50 ){
34                                          gamezone.getRodStatus()[gamezone.
                                                getSide() == Sides.UP ? 1 : 0].
                                                put(gamezone.rodPosition,
                                                RodStatus.NORMAL);
35                                          i = 0;
36                                          gamezone.setToNormalRod(null);
37                                          gamezone.setToNormalSide(null);
38                                  }else
39                                          i++;
```

1.3 Network

1.3.1 Client.java

```
1  public class Client {
2          private static Socket socketChat;
3          private static Socket socketPlayer;
4          private static Socket socketMatch;
5          private static Socket socketGame;
6          private ChatClient cc;
7          private PlayerClient pc;
8          private MatchClient mc;
```

```
9           private GameClient gc;
10          private Thread tChat;
11          private Thread tPlayer;
12          private Thread tMatch;
13          private Thread tGame;
14
15          private ClientBabyfoot main;
16          public Client(ClientBabyfoot m){
17                  main = m;
18
19                  Scanner sc = new Scanner(System.in);
20              try {
21                  Client.socketChat = new Socket("127.0.0.1",main.getPort());
22                  System.out.println("Connexion établie avec le serveur pour
                        le chat");
23                  cc = new ChatClient( Client.socketChat, main );
24                  tChat = new Thread(getCc());
25                  tChat.start();
26                  Client.socketPlayer = new Socket("127.0.0.1",main.getPort())
                        ;
27                  System.out.println("Connexion établie avec le serveur pour
                        les joueurs");
28                  pc = new PlayerClient( Client.socketPlayer, main );
29                  tPlayer = new Thread(pc);
30                  tPlayer.start();
31                  Client.socketMatch = new Socket("127.0.0.1",main.getPort());
32                  System.out.println("Connexion établie avec le serveur pour
                        les matchs");
33                  mc = new MatchClient( Client.socketMatch );
34                  tMatch = new Thread(mc);
35                  tMatch.start();
36                  Client.socketGame = new Socket("127.0.0.1",main.getPort());
37                  System.out.println("Connexion établie avec le serveur pour
                        les jeux");
38                  gc = new GameClient( Client.socketGame, main );
39                  tGame = new Thread(gc);
40                  tGame.start();
41              } catch (UnknownHostException e) {
42                System.err.println("Impossible de se connecter à l'adresse
                      127.0.0.1 ");
43              } catch (IOException e) {
44                System.err.println("Aucun serveur à l'écoute du port " + main.
                      getPort());
45              }
46              sc.close();
47          }
```

1.3.2 MatchClient.java

```
1  public class MatchClient implements Runnable {
2           private Socket socket;
3       private PrintWriter out = null;
4       public PrintWriter getOut() {
5                   return out;
6           }
7
8           private BufferedReader in = null;
9       private MatchReceptionMessage prc;
10      private boolean ok;
```

```
11      private String[] serverList;
12      private String[] matchDatas;
13      private int leftScore = 0;
14      private int rightScore = 0;
15      private boolean pause = false;
16      private boolean toDelete = false;
17      private int statusRod = 0;
18
19          public MatchClient(Socket s){
20          socket = s;
21      }
22
23      public void run() {
24          try {
25                  out = new PrintWriter(socket.getOutputStream());
26              in = new BufferedReader(new InputStreamReader(socket.
                    getInputStream()));
27              prc = new MatchReceptionMessage(in, this);
28              Thread tReceptionMessage = new Thread(prc);
29              tReceptionMessage.start();
30
31          } catch (IOException e) {
32              System.err.println("Le serveur distant s'est déconnecté !");
33          }
34      }
35
36          public void setRodPositions(String login, int[] rodPositions,
                RodStatus[] rodStatus) {
37                  out.println("match" + Utils.SEPARATOR + "setrod" + Utils.
                        SEPARATOR + login + Utils.SEPARATOR
38                                  + rodPositions[0] + Utils.SEPARATOR +
                                        rodPositions[1] + Utils.SEPARATOR +
                                        rodPositions[2] + Utils.SEPARATOR +
                                        rodPositions[3]
39                                  + Utils.SEPARATOR + rodStatus[0] + Utils.
                                        SEPARATOR + rodStatus[1] + Utils.
                                        SEPARATOR + rodStatus[2] + Utils.
                                        SEPARATOR + rodStatus[3]);
40          out.flush();
41          }
42
43          public String[] getServers() {
44                  out.println("match" + Utils.SEPARATOR + "getserverslist" );
45          out.flush();
46                  try {
47                          Thread.currentThread();
48                          Thread.sleep(100);
49                  } catch (InterruptedException e) {
50                          e.printStackTrace();
51                  }
52                  return getServerList();
53          }
54
55          public boolean setServerFromHost(String login, String loginHost) {
56                  out.println("player" + Utils.SEPARATOR + "joinmatch" + Utils
                        .SEPARATOR + login + Utils.SEPARATOR + loginHost );
57          out.flush();
58          try {
59                          Thread.currentThread();
60                          Thread.sleep(100);
```

```
61                  } catch (InterruptedException e) {
62                          e.printStackTrace();
63                  }
64          if( statusRod != 0 ){
65                  setOk(false);
66                  return true;
67          }else{
68                  return false;
69          }
70          }
71
72          public String[] getMatchInfo(String login) {
73                  out.println("match" + Utils.SEPARATOR + "getmatchinfo" +
                       Utils.SEPARATOR + login );
74          out.flush();
75                  try {
76                          Thread.currentThread();
77                          Thread.sleep(200);
78                  } catch (InterruptedException e) {
79                          e.printStackTrace();
80                  }
81                  return getMatchDatas();
82          }
83
84          public String[] getServerList() {
85                  return serverList;
86          }
87
88          public void setServerList(String[] serverList) {
89                  this.serverList = serverList;
90          }
91
92          public boolean isOk() {
93                  return ok;
94          }
95
96          public void setOk(boolean ok) {
97                  this.ok = ok;
98          }
99
100         public void setMatchDatas(String[] md) {
101                 switch( Integer.valueOf( md[1] ) ){
102                         case 1:
103                                 matchDatas = new String[4];
104                                 matchDatas[1] = "1";
105                                 if( md[2].equals(" ") ){
106                                         matchDatas[2] = "";
107                                         matchDatas[3] = md[3];
108                                 }else if( md[3].equals(" ") ){
109                                         matchDatas[2] = md[2];
110                                         matchDatas[3] = md[4];
111                                 }
112                                 break;
113                         case 2:
114                                 matchDatas = new String[6];
115                                 matchDatas[1] = "2";
116                                 if( md[2].equals(" ") ){
117                                         matchDatas[2] = "";
118                                         matchDatas[3] = "";
119                                         matchDatas[4] = md[3];
```

```
120                                     matchDatas[5] = md[4];
121                             }else if( md[3].equals(" ") ){
122                                     matchDatas[2] = md[2];
123                                     matchDatas[3] = "";
124                                     matchDatas[4] = md[4];
125                                     matchDatas[5] = md[5];
126                             }else if( md[4].equals(" ") ){
127                                     matchDatas[2] = md[2];
128                                     matchDatas[3] = md[3];
129                                     matchDatas[4] = md[5];
130                                     matchDatas[5] = md[6];
131                             }else if( md[5].equals(" ") ){
132                                     matchDatas[2] = md[2];
133                                     matchDatas[3] = md[3];
134                                     matchDatas[4] = md[4];
135                                     matchDatas[5] = "";
136                             }else{
137                                     matchDatas[2] = md[2];
138                                     matchDatas[3] = md[3];
139                                     matchDatas[4] = md[4];
140                                     matchDatas[5] = md[5];
141                             }
142                             break;
143                     case 3:
144                             matchDatas = new String[5];
145                             matchDatas[1] = "3";
146                             if( md[2].equals(" ") ){
147                                     matchDatas[2] = "";
148                                     matchDatas[3] = md[3];
149                                     matchDatas[4] = md[4];
150                             }else if( md[3].equals(" ") ){
151                                     matchDatas[2] = md[2];
152                                     matchDatas[3] = md[4];
153                                     matchDatas[4] = md[5];
154                             }
155                             break;
156             }
157             matchDatas[0] = md[0];
158     }
159
160     public String[] getMatchDatas() {
161             return matchDatas;
162     }
163
164     public void initMatchDatas(int i) {
165             matchDatas = new String[i];
166     }
167
168     public void runMatch(String login) {
169             out.println("match" + Utils.SEPARATOR + "run" + Utils.
                    SEPARATOR + login );
170     out.flush();
171             try {
172                     Thread.currentThread();
173                     Thread.sleep(200);
174             } catch (InterruptedException e) {
175                     e.printStackTrace();
176             }
177     }
178
```

```
179         public void stopMatch(String login) {
180                 out.println("match" + Utils.SEPARATOR + "stop" + Utils.
                        SEPARATOR + login );
181         out.flush();
182         statusRod = 0;
183         }
184 
185         public void deleteMatch() {
186                 toDelete = true;
187         }
188 
189 
190     public boolean isToDelete() {
191                 return toDelete;
192         }
193 
194         public void setToDelete(boolean toDelete) {
195                 this.toDelete = toDelete;
196         }
197 
198         public void sendShoot(String login, long duration, Rod rodPosition,
                Sides side) {
199                 out.println("match" + Utils.SEPARATOR + "shoot" + Utils.
                        SEPARATOR + login + Utils.SEPARATOR + rodPosition + Utils
                        .SEPARATOR + side );
200         out.flush();
201         }
202 
203         public void quitMatch(String login) {
204                 out.println("match" + Utils.SEPARATOR + "quit" + Utils.
                        SEPARATOR + login );
205         out.flush();
206         }
207 
208         public void setMatchInfos(String message) {
209                 if( message != null ){
210                         String[] s = message.split(Utils.SEPARATOR);
211                         setLeftScore(Integer.valueOf(s[1]));
212                         setRightScore(Integer.valueOf(s[2]));
213                         setPause(Boolean.valueOf(s[3]));
214                 }
215         }
216 
217         public void setMatchCarac(String message) {
218                 try{
219                         statusRod = Integer.valueOf(message);
220                 }catch(NumberFormatException e){
221                 }
222         }
223 
224         public int getLeftScore() {
225                 return leftScore;
226         }
227 
228         public void setLeftScore(int leftScore) {
229                 this.leftScore = leftScore;
230         }
231 
232         public int getRightScore() {
233                 return rightScore;
```

```
234         }
235
236         public void setRightScore(int rightScore) {
237                 this.rightScore = rightScore;
238         }
239
240         public boolean isPause() {
241                 return pause;
242         }
243
244         public void setPause(boolean pause) {
245                 this.pause = pause;
246         }
247
248         public int getStatusRod() {
249                 return statusRod;
250         }
251
252         public void setStatusRod(int statusRod) {
253                 this.statusRod = statusRod;
254         }
255
256         public boolean askForPause(String login, boolean currentState) {
257                 out.println("match" + Utils.SEPARATOR + ( currentState ? "
                    startagain" : "askforpause" ) + Utils.SEPARATOR + login )
                    ;
258         out.flush();
259                 try {
260                         Thread.currentThread();
261                         Thread.sleep(100);
262                 } catch (InterruptedException e) {
263                         e.printStackTrace();
264                 }
265                 if( ok ){
266                         setOk(false);
267                 return true;
268                 }
269                 else
270                         return false;
271         }
272 }
273
274 class MatchReceptionMessage implements Runnable{
275
276         BufferedReader in;
277         boolean ready;
278         MatchClient matchClient;
279
280         public MatchReceptionMessage(BufferedReader in, MatchClient
                matchClient){
281                 this.in = in;
282                 this.matchClient = matchClient;
283         }
284
285         public void run() {
286                 int mode = 0;
287                 int n = 0;
288                 int type = 0;
289                 System.out.println("Prêt à la réception pour les match !");
290                 try {
```

```
291                         while(true){
292                 String message = in.readLine();
293                 if(message.equals("matchinfo" + Utils.SEPARATOR + "deleted")
                        ){
294                         matchClient.deleteMatch();
295                 }
296                 if( ( message.equals("matchlist" + Utils.SEPARATOR + "
                        beginning") || message.equals("matchdata" + Utils.
                        SEPARATOR + "beginning") ) && mode == 0 ){
297                         if( message.equals("matchdata" + Utils.SEPARATOR + "
                            beginning") ){
298                                 type = 1;
299                                 matchClient.initMatchDatas(5);
300                                 mode = 2;
301                         }else
302                                 mode = 1;
303                 }else if( mode == 1 ){
304                         if( type == 0){
305                                 matchClient.setServerList(new String[Integer
                                    .valueOf(message)]);
306                         }
307                         mode = 2;
308                 }else if( ( message.equals( "matchlist" + Utils.SEPARATOR +
                        "end") || message.equals("matchdata" + Utils.SEPARATOR +
                        "end") )  && mode == 2 ){
309                         mode = 0;
310                         type = 0;
311                         n = 0;
312                 }else if( mode == 2 ){
313                         if( type == 0)
314                                 matchClient.getServerList()[n++] = message;
315                         else if( type == 1){
316                                 matchClient.setMatchDatas(message.split(
                                    Utils.SEPARATOR));
317                         }
318                 }else{
319                         matchClient.setOk(message.equals("true"));
320                         matchClient.setMatchCarac(message);
321                 }
322                         }
323         } catch (IOException e) {
324             e.printStackTrace();
325         }
326         }
```

1.4 Classe de test

```
1  public class Tester {
2          public static void main(String[] args) throws IOException{
3                  ServerBabyfoot s = new ServerBabyfoot(new ServerSocket(2010)
                       );
4                  Thread serveur = new Thread(s);
5                  serveur.start();
6
7                  ClientBabyfoot m1 = new ClientBabyfoot(2010);
8                  Thread t1 = new Thread(new TesterThread(m1));
9                  t1.start();
10         }
```

```
11 }
12
13 class TesterThread implements Runnable {
14         private ClientBabyfoot main;
15         public TesterThread(ClientBabyfoot m){
16                 main = m;
17         }
18         public void run(){
19                 main.init();
20                 String loginHost = "bob";
21                 ((ConnexionPanel) main.actualPanel).logIn( loginHost );
22                 ((MenuPanel) main.actualPanel).getWindow().setContentPane(
                       new NewPanel(((MenuPanel) main.actualPanel).getWindow()))
                       ;
23                 ((NewPanel) main.actualPanel).getWindow().setVisible(true);
24                 ((NewPanel) main.actualPanel).getWindow().getMain().
                       getPlayer().addMatch(1);
25                 ((NewPanel) main.actualPanel).getWindow().getMain().
                       getPlayer().setBoss(true);
26                 ((NewPanel) main.actualPanel).getWindow().setContentPane(new
                        WaitingRoomPanel(((NewPanel) main.actualPanel).getWindow
                       ()));
27                 ((WaitingRoomPanel) main.actualPanel).getWindow().setVisible
                       (true);
28                 PrintWriter out = ((WaitingRoomPanel) main.actualPanel).
                       getWindow().getMain().getClient().getMc().getOut();
29                 String login = "coucou";
30                 out.println("player" + Utils.SEPARATOR + "add" + Utils.
                       SEPARATOR + login);
31                 out.flush();
32                 out.println("player" + Utils.SEPARATOR + "joinmatch" + Utils
                       .SEPARATOR + login + Utils.SEPARATOR + loginHost );
33                 out.flush();
34                 out.println("match" + Utils.SEPARATOR + "run" + Utils.
                       SEPARATOR + loginHost );
35         out.flush();
36                 ((WaitingRoomPanel) main.actualPanel).getWindow().getMain().
                       getClient().getGc().startThread();
37                 ((WaitingRoomPanel) main.actualPanel).getWindow().
                       setContentPane(new GamePanel(((WaitingRoomPanel) main.
                       actualPanel).getWindow(),false));
38                 ((WaitingRoomPanel) main.actualPanel).getWindow().
                       getContentPane().requestFocusInWindow();
39                 ((WaitingRoomPanel) main.actualPanel).getWindow().setVisible
                       (true);
40         }
41 }
```

Chapitre 2

Serveur

2.1 Core

2.1.1 Fonction de tir dans Match.java

```
1         public void shoot(String r, String s) {
2                 Sides side = Sides.valueOf(s);
3                 Rod rod = Rod.valueOf(r);
4                 if( side == Sides.DOWN ){
5                         if( rod == Rod.GARDIEN ){
6                                 int yPosition = Utils.getYPositionPlayer(
                                      yRodPositions, Rod.GARDIEN, 1, 1, side );
7                                 if( ballX >= Utils.GARDIEN_POSITION && ballX
                                       <= Utils.GARDIEN_POSITION + Utils.
                                      IMAGE_PLAYER_SHOOTING_X + Utils.
                                      BALL_RADIUS
8                                                 //&& ballSpeedX <= 0
9                                                 && ballY >= yPosition &&
                                                     ballY <= Utils.
                                                     IMAGE_PLAYER_Y +
                                                     yPosition ){
10                                        ballSpeedX = Utils.MAX_INITIAL_SPEED
                                              ;
11                                        ballSpeedY *= -0.1;
12                                }
13                        }else if( rod == Rod.DEFENSE ){
14                                boolean test = false;
15                                for( int i = 1; i <= 2; i++ ){
16                                        int yPosition = Utils.
                                             getYPositionPlayer(yRodPositions,
                                              Rod.DEFENSE, i, 2, side );
17                                        test |= ( ballY >= yPosition &&
                                             ballY <= Utils.IMAGE_PLAYER_Y +
                                             yPosition );
18                                        if( test ) break;
19                                }
20                                if( ballX >= Utils.DEFENSE_POSITION && ballX
                                       <= Utils.DEFENSE_POSITION + Utils.
                                      IMAGE_PLAYER_SHOOTING_X + Utils.
                                      BALL_RADIUS
21                                                //&& ballSpeedX <= 0
22                                                && test ){
23                                        ballSpeedX = Utils.MAX_INITIAL_SPEED
                                              ;
24                                        ballSpeedY *= -0.1;
```

```
25                              }
26                          }else if( rod == Rod.MILIEU ){
27                              boolean test = false;
28                              for( int i = 1; i <= 5; i++ ){
29                                  int yPosition = Utils.
                                        getYPositionPlayer(yRodPositions,
                                         Rod.MILIEU, i, 5, side );
30                                  test |= ( ballY >= yPosition &&
                                        ballY <= Utils.IMAGE_PLAYER_Y +
                                        yPosition );
31                                  if( test ) break;
32                              }
33                              if( ballX >= Utils.MILIEU_POSITION && ballX
                                    <= Utils.MILIEU_POSITION + Utils.
                                    IMAGE_PLAYER_SHOOTING_X + Utils.
                                    BALL_RADIUS
34                                          //&& ballSpeedX <= 0
35                                          && test ){
36                                  ballSpeedX = Utils.MAX_INITIAL_SPEED
                                        ;
37                                  ballSpeedY *= -0.1;
38                              }
39                          }else if( rod == Rod.ATTAQUE ){
40                              boolean test = false;
41                              for( int i = 1; i <= 3; i++ ){
42                                  int yPosition = Utils.
                                        getYPositionPlayer(yRodPositions,
                                         Rod.ATTAQUE, i, 3, side );
43                                  test |= ( ballY >= yPosition - Utils
                                        .BALL_RADIUS && ballY <= Utils.
                                        BALL_RADIUS + Utils.
                                        IMAGE_PLAYER_Y + yPosition );
44                                  if( test ) break;
45                              }
46                              if( ballX >= Utils.ATTAQUE_POSITION && ballX
                                     <= Utils.ATTAQUE_POSITION + Utils.
                                    IMAGE_PLAYER_SHOOTING_X + Utils.
                                    BALL_RADIUS
47                                          //&& ballSpeedX <= 0
48                                          && test ){
49                                  ballSpeedX = Utils.MAX_INITIAL_SPEED
                                        ;
50                                  ballSpeedY *= -0.1;
51                              }
52                          }
53                      }else{
54                          if( rod == Rod.GARDIEN ){
55                              int yPosition = Utils.getYPositionPlayer(
                                    yRodPositions, Rod.GARDIEN, 1, 1, side);
56                              if( ballX + Utils.IMAGE_PLAYER_SHOOTING_X +
                                    Utils.BALL_RADIUS >= Utils.
                                    GARDIEN_POSITION  && ballX <= Utils.
                                    GARDIEN_POSITION
57                                          //&& ballSpeedX <= 0
58                                          && ballY >= yPosition &&
                                                ballY <= Utils.
                                                IMAGE_PLAYER_Y +
                                                yPosition ){
59                                  ballSpeedX = Utils.MAX_INITIAL_SPEED
                                        ;
```

```
60                                  ballSpeedY *= -0.1;
61                              }
62                          }else if( rod == Rod.DEFENSE ){
63                              boolean test = false;
64                              for( int i = 1; i <= 2; i++ ){
65                                  int yPosition = Utils.
                                        getYPositionPlayer(yRodPositions,
                                         Rod.DEFENSE, i, 2, side);
66                                  test |= ( ballY >= yPosition &&
                                        ballY <= Utils.IMAGE_PLAYER_Y +
                                        yPosition );
67                                  if( test ) break;
68                              }
69                              if( ballX + Utils.IMAGE_PLAYER_SHOOTING_X +
                                    Utils.BALL_RADIUS >= Utils.
                                    DEFENSE_POSITION && ballX <= Utils.
                                    DEFENSE_POSITION
70                                          //&& ballSpeedX <= 0
71                                          && test ){
72                                  ballSpeedX = Utils.MAX_INITIAL_SPEED
                                        ;
73                                  ballSpeedY *= -0.1;
74                              }
75                          }else if( rod == Rod.MILIEU ){
76                              boolean test = false;
77                              for( int i = 1; i <= 5; i++ ){
78                                  int yPosition = Utils.
                                        getYPositionPlayer(yRodPositions,
                                         Rod.MILIEU, i, 5, side);
79                                  test |= ( ballY >= yPosition &&
                                        ballY <= Utils.IMAGE_PLAYER_Y +
                                        yPosition );
80                                  if( test ) break;
81                              }
82                              if( ballX + Utils.IMAGE_PLAYER_SHOOTING_X +
                                    Utils.BALL_RADIUS >= Utils.
                                    MILIEU_POSITION && ballX <= Utils.
                                    MILIEU_POSITION
83                                          //&& ballSpeedX <= 0
84                                          && test ){
85                                  ballSpeedX = Utils.MAX_INITIAL_SPEED
                                        ;
86                                  ballSpeedY *= -0.1;
87                              }
88                          }else if( rod == Rod.ATTAQUE ){
89                              boolean test = false;
90                              for( int i = 1; i <= 3; i++ ){
91                                  int yPosition = Utils.
                                        getYPositionPlayer(yRodPositions,
                                         Rod.ATTAQUE, i, 3, side);
92                                  test |= ( ballY >= yPosition - Utils
                                        .BALL_RADIUS && ballY <= Utils.
                                        BALL_RADIUS + Utils.
                                        IMAGE_PLAYER_Y + yPosition );
93                                  if( test ) break;
94                              }
95                              if( ballX + Utils.IMAGE_PLAYER_SHOOTING_X +
                                    Utils.BALL_RADIUS >= Utils.
                                    ATTAQUE_POSITION && ballX <= Utils.
                                    ATTAQUE_POSITION
```

```
96                                                  //&& ballSpeedX <= 0
97                                                  && test ){
98                                              ballSpeedX = Utils.MAX_INITIAL_SPEED
                                                    ;
99                                              ballSpeedY *= -0.1;
100                                         }
101                                 }
102                         }
103                         rodStatus[side == Sides.UP ? 1 : 0].put(rod, RodStatus.
                                SHOOTING);
104         }
```

2.1.2 Thread de Match.java

```
1   class RefreshBallPosition implements Runnable {
2           private Match match;
3           private boolean run;
4           public RefreshBallPosition(Match m){
5                   match = m;
6                   setRun(true);
7           }
8
9           @Override
10          public void run() {
11                  try{
12                          while(isRun()){
13                                  if( !match.isPause() ){
14                                          //Si on a atteint le bord extérieur
                                               gauche et que la vitesse est bien
                                                négative (donc vers la gauche),
                                               on change de vitesse.
15                                          if( ( match.getBallX() -5 - Utils.
                                               BALL_RADIUS ) <=  ( Utils.
                                               GAP_EDGE + Utils.LINE_STRENGTH )
                                                ) {
16                                                  if( match.getBallSpeedX() <
                                                        0 )
17                                                          match.setBallSpeedX
                                                            ((-1)*match.
                                                            getBallSpeedX()+(
                                                             match.isSlow() ?
                                                             0 : match.
                                                            getBallSpeedX()/
                                                            Math.abs(match.
                                                            getBallSpeedX()))
                                                            );
18                                                  match.verifGoal();
19                                          //Si on a atteint le bord extérieur
                                               droit et que la vitesse est bien
                                               positive (donc vers la droite),
                                               on change de vitesse
20                                          }else if( ( match.getBallX()+5 ) >=
                                                ( Utils.WIDTH - Utils.GAP_EDGE -
                                                Utils.LINE_STRENGTH - Utils.
                                               BALL_RADIUS ) ){
21                                                  if( match.getBallSpeedX() >
                                                        0 )
22                                                          match.setBallSpeedX
                                                            ((-1)*match.
```

```
                                getBallSpeedX()+(
                                 match.isSlow() ?
                                 0 : match.
                                getBallSpeedX()/
                                Math.abs(match.
                                getBallSpeedX()))
                                );
23                      match.verifGoal();
24              }
25              //Si on a atteint le bord extérieur
                    haut et que la vitesse est bien n
                    égative (donc vers le haut), on
                    change de vitesse.
26              if( ( match.getBallY()-5 - Utils.
                    BALL_RADIUS  ) <=  ( Utils.
                    GAP_EDGE + Utils.LINE_STRENGTH )
                    ){
27                      if( match.getBallSpeedY() <
                            0 )
28                              match.setBallSpeedY
                                    ((-1)*match.
                                    getBallSpeedY()+(
                                     match.isSlow() ?
                                     0 : match.
                                    getBallSpeedY()/
                                    Math.abs(match.
                                    getBallSpeedY()))
                                    );
29              //Si on a atteint le bord extérieur
                    bas et que la vitesse est bien
                    positive (donc vers le bas), on
                    change de vitesse
30              }else if( ( match.getBallY() +5+
                    Utils.BALL_RADIUS  ) >=  ( Utils.
                    HEIGHT - Utils.GAP_EDGE - Utils.
                    LINE_STRENGTH ) ) {
31                      if( match.getBallSpeedY() >
                            0 )
32                              match.setBallSpeedY
                                    ((-1)*match.
                                    getBallSpeedY()+(
                                     match.isSlow() ?
                                     0 : match.
                                    getBallSpeedY()/
                                    Math.abs(match.
                                    getBallSpeedY()))
                                    );
33              }
34
35              CollisionType resultatsCollisions =
                    match.testCollisions();
36              if( resultatsCollisions ==
                    CollisionType.SIDES )
37                      match.setBallSpeedX((-1)*
                            match.getBallSpeedX()+(
                            match.isSlow() ? 0 : 1*
                            match.getBallSpeedX()/11)
                            );
38              else if( resultatsCollisions ==
                    CollisionType.UPANDDOWN )
```

```
39                                                  match.setBallSpeedY((-1)*
                                                        match.getBallSpeedY()+(
                                                        match.isSlow() ? 0 : 1*
                                                        match.getBallSpeedY()/11)
                                                        );
40
41                                              match.addBallX(match.getBallSpeedX()
                                                    );
42                                              match.addBallY(match.getBallSpeedY()
                                                    );
43                                              Thread.sleep(25);
44                                      }
45                              }
46                      }catch(InterruptedException e){
47                              e.printStackTrace();
48                      }
49              }
50
51              public boolean isRun() {
52                      return run;
53              }
54
55              public void setRun(boolean run) {
56                      this.run = run;
57              }
58      }
```

2.1.3 Collisions.java

```
1   public class Collisions {
2
3           private float ballX;
4           private float ballY;
5           private float ballSpeedX;
6           private float ballSpeedY;
7
8           private Hashtable<Sides, Hashtable<Rod, Long>> lastCollision;
9
10          @SuppressWarnings("unchecked")
11          private Hashtable<Rod, Integer>[] rodPositions = new Hashtable[2] ;
12
13          public Collisions() {
14                  lastCollision = new Hashtable<Sides, Hashtable<Rod, Long>>()
                        ;
15                  Hashtable<Rod, Long> tUp = new Hashtable<Rod, Long>();
16                  Hashtable<Rod, Long> tDown = new Hashtable<Rod, Long>();
17
18                  lastCollision.put(Sides.UP, tUp);
19                  lastCollision.put(Sides.DOWN, tDown);
20                  zeroLast();
21          }
22
23          private void zeroLast(){
24                  lastCollision.get(Sides.UP).put(Rod.GARDIEN, 0L);
25                  lastCollision.get(Sides.UP).put(Rod.DEFENSE, 0L);
26                  lastCollision.get(Sides.UP).put(Rod.MILIEU, 0L);
27                  lastCollision.get(Sides.UP).put(Rod.ATTAQUE, 0L);
28                  lastCollision.get(Sides.DOWN).put(Rod.GARDIEN, 0L);
29                  lastCollision.get(Sides.DOWN).put(Rod.DEFENSE, 0L);
```

```
30                  lastCollision.get(Sides.DOWN).put(Rod.MILIEU, 0L);
31                  lastCollision.get(Sides.DOWN).put(Rod.ATTAQUE, 0L);
32          }
33
34          public CollisionType testCollisions(Integer position, Rod rod){
35                  CollisionType rodBottom = null;
36                  CollisionType rodTop = null;
37                  rodTop = testCollisionsTop(position, rod);
38                  rodBottom = testCollisionsBottom(position, rod);
39                  if( rodTop != null || rodBottom != null ){
40                          Sides side = ( rodTop != null ? Sides.UP : Sides.
                               DOWN );
41                          Sides otherSide = ( rodTop != null ? Sides.DOWN :
                               Sides.UP );
42                          if( (lastCollision.get(side)).get(rod) > 0L &&
                               System.currentTimeMillis() - (lastCollision.get(
                               side)).get(rod) < 500 ){
43                                  if( (lastCollision.get(otherSide)).get(rod)
                                       > System.currentTimeMillis() - 500 ) {
44                                          (lastCollision.get(otherSide)).put(
                                               rod,System.currentTimeMillis());
45                                  }else{
46                                          //System.out.println("Avortée !" +
                                               side + " - " + rod);
47                                          return null;
48                                  }
49                                  //System.out.println("Avortée - " + side + "
                                       - " + rod);
50                                  return null;
51                          }
52                          else{
53                                  //zeroLast();
54                                  (lastCollision.get(side)).put(rod,System.
                                       currentTimeMillis());
55                          }//*/
56                  }
57                  return ( rodTop == null ? rodBottom : rodTop );
58          }
59
60          public CollisionType testCollisionsTop(Integer position, Rod rod){
61                  //y final : y + i*h/(1+nb)-Utils.IMAGE_PLAYER_Y/2 + yDecal[
                       rightPlayer ? 1 : 0].get(rod)-Utils.Y_STAGGERING_DEFAULT.
                       get(rod)
62                  // i va de 1 à nb où nb est le nombre de joueurs sur une
                       barre
63                  float yTopHitBox = position + Utils.GAP_EDGE;
64                  float xLeftHitBox = 0;
65                  CollisionType ballPosition = null;
66                  switch(rod){
67                  //Utils.WIDTH-Utils.LINE_STRENGTH-Utils.GARDIEN_POSITION-
                       Utils.IMAGE_PLAYER_X/3
68                          case GARDIEN:
69                                  //yTopHitBox = position + Utils.HEIGHT/2-
                                       Utils.IMAGE_PLAYER_Y/2 - Utils.
                                       Y_STAGGERING_DEFAULT.get(Rod.GARDIEN);
70                                  yTopHitBox = Utils.getYPositionPlayer(
                                       position, rod, 1, 1 );
71                                  xLeftHitBox = Utils.WIDTH - (Utils.
                                       GARDIEN_POSITION+(float)(Utils.
                                       IMAGE_PLAYER_X/3));
```

```
72                              ballPosition = isBallInCollision(
                                    xLeftHitBox, yTopHitBox, xLeftHitBox +
                                    Utils.IMAGE_PLAYER_X, yTopHitBox + Utils.
                                    IMAGE_PLAYER_Y );
73                              if( ballPosition != null )
74                                      return ballPosition;
75                              break;
76                      case DEFENSE:
77                              //yTopHitBox = position + Utils.GAP_EDGE +
                                    Utils.HEIGHT/3-Utils.IMAGE_PLAYER_Y/2 -
                                    Utils.Y_STAGGERING_DEFAULT.get(Rod.
                                    DEFENSE);
78                              yTopHitBox = Utils.getYPositionPlayer(
                                    position, rod, 1, 2 );
79                              xLeftHitBox = Utils.WIDTH - (Utils.
                                    DEFENSE_POSITION+(float)(Utils.
                                    IMAGE_PLAYER_X/3));
80                              ballPosition = isBallInCollision(
                                    xLeftHitBox, yTopHitBox, xLeftHitBox +
                                    Utils.IMAGE_PLAYER_X, yTopHitBox + Utils.
                                    IMAGE_PLAYER_Y );
81                              if( ballPosition != null )
82                                      return ballPosition;
83                              //yTopHitBox = position + 2*Utils.HEIGHT/3-(
                                    float)(Utils.IMAGE_PLAYER_Y/2) - Utils.
                                    Y_STAGGERING_DEFAULT.get(Rod.DEFENSE);
84                              ballPosition = isBallInCollision(
                                    xLeftHitBox, yTopHitBox, xLeftHitBox +
                                    Utils.IMAGE_PLAYER_X, yTopHitBox + Utils.
                                    IMAGE_PLAYER_Y );
85                              if( ballPosition != null )
86                                      return ballPosition;
87                              break;
88                      case MILIEU:
89                              xLeftHitBox = Utils.WIDTH - (Utils.
                                    MILIEU_POSITION+(float)(Utils.
                                    IMAGE_PLAYER_X/3));
90                              for( int i = 1; i < 6; i++ ){
91                                      //yTopHitBox = position + i*Utils.
                                            HEIGHT/6-(float)(Utils.
                                            IMAGE_PLAYER_Y/2) - Utils.
                                            Y_STAGGERING_DEFAULT.get(Rod.
                                            MILIEU);
92                                      yTopHitBox = Utils.
                                            getYPositionPlayer( position, rod
                                            , i, 5 );
93                                      ballPosition = isBallInCollision(
                                            xLeftHitBox, yTopHitBox,
                                            xLeftHitBox + Utils.
                                            IMAGE_PLAYER_X, yTopHitBox +
                                            Utils.IMAGE_PLAYER_Y );
94                                      if( ballPosition != null )
95                                              return ballPosition;
96                              }
97                              break;
98                      case ATTAQUE:
99                              xLeftHitBox = Utils.WIDTH - (Utils.
                                    ATTAQUE_POSITION+(float)(Utils.
                                    IMAGE_PLAYER_X/3));
100                             for( int i = 1; i < 4; i++ ){
```

```
101                                     //yTopHitBox = position + i*Utils.
                                            HEIGHT/4-(float)(Utils.
                                            IMAGE_PLAYER_Y/2) - Utils.
                                            Y_STAGGERING_DEFAULT.get(Rod.
                                            ATTAQUE);
102                                     yTopHitBox = Utils.
                                            getYPositionPlayer( position, rod
                                            , i, 3 );
103                                     ballPosition = isBallInCollision(
                                            xLeftHitBox, yTopHitBox,
                                            xLeftHitBox + Utils.
                                            IMAGE_PLAYER_X, yTopHitBox +
                                            Utils.IMAGE_PLAYER_Y );
104                                     if( ballPosition != null )
105                                             return ballPosition;
106                             }
107                             break;
108                     }
109                     return null;
110             }
111
112             public CollisionType testCollisionsBottom(Integer position, Rod rod)
                        {
113                     float yTopHitBox = position + Utils.GAP_EDGE;
114                     float xLeftHitBox = 0;
115                     CollisionType ballPosition = null;
116                     switch(rod){
117                             case GARDIEN:
118                                     //yTopHitBox = position + Utils.HEIGHT/2-
                                            Utils.IMAGE_PLAYER_Y/2 - Utils.
                                            Y_STAGGERING_DEFAULT.get(Rod.GARDIEN);
119                                     yTopHitBox = Utils.getYPositionPlayer(
                                            position, rod, 1, 1 );
120                                     xLeftHitBox = Utils.GARDIEN_POSITION-(float)
                                            (Utils.IMAGE_PLAYER_X/3);
121                                     ballPosition = isBallInCollision(
                                            xLeftHitBox, yTopHitBox, xLeftHitBox +
                                            Utils.IMAGE_PLAYER_X, yTopHitBox + Utils.
                                            IMAGE_PLAYER_Y );
122                                     if( ballPosition != null )
123                                             return ballPosition;
124                                     break;
125                             case DEFENSE:
126                                     //yTopHitBox = position + Utils.HEIGHT/3-
                                            Utils.IMAGE_PLAYER_Y/2 - Utils.
                                            Y_STAGGERING_DEFAULT.get(Rod.DEFENSE);
127                                     yTopHitBox = Utils.getYPositionPlayer(
                                            position, rod, 1, 2 );
128                                     xLeftHitBox = Utils.DEFENSE_POSITION-(float)
                                            (Utils.IMAGE_PLAYER_X/3);
129                                     ballPosition = isBallInCollision(
                                            xLeftHitBox, yTopHitBox, xLeftHitBox +
                                            Utils.IMAGE_PLAYER_X, yTopHitBox + Utils.
                                            IMAGE_PLAYER_Y );
130                                     if( ballPosition != null )
131                                             return ballPosition;
132                                     //yTopHitBox = position + 2*Utils.HEIGHT/3-
                                            Utils.IMAGE_PLAYER_Y/2 - Utils.
                                            Y_STAGGERING_DEFAULT.get(Rod.DEFENSE);
```

```
133                                 yTopHitBox = Utils.getYPositionPlayer(
                                        position, rod, 2, 2 );
134                                 ballPosition = isBallInCollision(
                                        xLeftHitBox, yTopHitBox, xLeftHitBox +
                                        Utils.IMAGE_PLAYER_X, yTopHitBox + Utils.
                                        IMAGE_PLAYER_Y );
135                                 if( ballPosition != null )
136                                         return ballPosition;
137                                 break;
138                         case MILIEU:
139                                 xLeftHitBox = Utils.MILIEU_POSITION-(float)(
                                        Utils.IMAGE_PLAYER_X/3);
140                                 for( int i = 1; i < 6; i++ ){
141                                         //yTopHitBox = position + i*Utils.
                                            HEIGHT/6-Utils.IMAGE_PLAYER_Y/2 -
                                             Utils.Y_STAGGERING_DEFAULT.get(
                                            Rod.MILIEU);
142                                         yTopHitBox = Utils.
                                            getYPositionPlayer( position, rod
                                            , i, 5 );
143                                         ballPosition = isBallInCollision(
                                            xLeftHitBox, yTopHitBox,
                                            xLeftHitBox + Utils.
                                            IMAGE_PLAYER_X, yTopHitBox +
                                            Utils.IMAGE_PLAYER_Y );
144                                         if( ballPosition != null )
145                                                 return ballPosition;
146                                 }
147                                 break;
148                         case ATTAQUE:
149                                 xLeftHitBox = Utils.ATTAQUE_POSITION-(float)
                                        (Utils.IMAGE_PLAYER_X/3);
150                                 for( int i = 1; i < 4; i++ ){
151                                         //yTopHitBox = position + i*Utils.
                                            HEIGHT/4-Utils.IMAGE_PLAYER_Y/2 -
                                             Utils.Y_STAGGERING_DEFAULT.get(
                                            Rod.ATTAQUE);
152                                         yTopHitBox = Utils.
                                            getYPositionPlayer( position, rod
                                            , i, 3 );
153                                         ballPosition = isBallInCollision(
                                            xLeftHitBox, yTopHitBox,
                                            xLeftHitBox + Utils.
                                            IMAGE_PLAYER_X, yTopHitBox +
                                            Utils.IMAGE_PLAYER_Y );
154                                         if( ballPosition != null )
155                                                 return ballPosition;
156                                 }
157                                 break;
158                 }
159                 return null;
160         }
161
162         public CollisionType isBallInCollision(float xLeftTop, float
                yLeftTop, float xRightBottom, float yRightBottom ){
163                 /*if( ( Math.pow( ballX - xLeftTop, 2) + Math.pow( ballY -
                        yLeftTop, 2 ) ) <= Math.pow( Utils.BALL_RADIUS, 2 )
164                                 || ( Math.pow( ballX - xRightBottom, 2) +
                                    Math.pow( ballY - yRightBottom, 2 ) ) <=
                                    Math.pow( Utils.BALL_RADIUS, 2 ) ){
```

```
165                         if( Math.abs(ballX - Utils.BALL_RADIUS - xLeftTop) /
                                Utils.IMAGE_PLAYER_X <= Math.abs(ballY - Utils.
                               BALL_RADIUS  - yLeftTop) / Utils.IMAGE_PLAYER_Y )
                                return CollisionType.UPANDDOWN;
166                         else return CollisionType.SIDES;
167                 }*/
168
169                 //On se trouve dans le coin de droite en bas.
170                 if( ballX - Utils.BALL_RADIUS <= xRightBottom && ballX -
                       Utils.BALL_RADIUS >= xLeftTop
171                                 && ballY - Utils.BALL_RADIUS <= yRightBottom
                                        && ballY - Utils.BALL_RADIUS >= yLeftTop
                                        )
172                         return verifWhichSide( ballX, ballY, xRightBottom,
                               yRightBottom ) ? CollisionType.UPANDDOWN :
                               CollisionType.SIDES;
173                 //On se trouve dans le coin de droite en haut.
174                 else if( ballX - Utils.BALL_RADIUS <= xRightBottom && ballX
                       - Utils.BALL_RADIUS >= xLeftTop
175                                         && ballY + Utils.BALL_RADIUS <=
                                               yRightBottom && ballY + Utils.
                                               BALL_RADIUS >= yLeftTop )
176                         return verifWhichSide( ballX, ballY, xRightBottom,
                               yLeftTop ) ? CollisionType.UPANDDOWN :
                               CollisionType.SIDES;
177                 //On se trouve dans le coin de gauche en haut.
178                 else if ( ballX + Utils.BALL_RADIUS <= xRightBottom && ballX
                        + Utils.BALL_RADIUS >= xLeftTop
179                                         && ballY + Utils.BALL_RADIUS <=
                                               yRightBottom && ballY + Utils.
                                               BALL_RADIUS >= yLeftTop )
180                         return verifWhichSide( ballX, ballY, xLeftTop,
                               yLeftTop ) ? CollisionType.UPANDDOWN :
                               CollisionType.SIDES;
181                 //On se trouve dans le coin de gauche en bas.
182                 else if ( ballX + Utils.BALL_RADIUS <= xRightBottom && ballX
                        + Utils.BALL_RADIUS >= xLeftTop
183                                         && ballY - Utils.BALL_RADIUS <=
                                               yRightBottom && ballY - Utils.
                                               BALL_RADIUS >= yLeftTop )
184                         return verifWhichSide( ballX, ballY, xLeftTop,
                               yRightBottom ) ? CollisionType.UPANDDOWN :
                               CollisionType.SIDES;
185                 return null;
186         }
187
188         public boolean verifWhichSide( float xb, float yb, float xc, float
                yc )
189         {
190                 return ( Math.abs(( xb - xc ) / ( yb - yc )) <= 1 );
191         }
192
193         public void setBallPosition(float ballX, float ballY, float
                ballSpeedX, float ballSpeedY) {
194                 this.ballX = ballX;
195                 this.ballY = ballY;
196                 this.ballSpeedX = ballSpeedX;
197                 this.ballSpeedY = ballSpeedY;
198         }
199 }
```

2.2 Network

2.2.1 PlayerServer.java

```
1  public class PlayerServer extends AbstractServer {
2          private HashMap<String, Player> liste = new HashMap<String, Player
               >();
3
4          public void handle(BufferedReader in, PrintWriter out){
5                  String[] datas = query.split(Utils.SEPARATOR);
6                  String task = datas[1];
7          String login = datas[2];
8          if( task.equals("add") ){
9                  if( addPlayer(login) )
10                         out.println("true");
11                 else
12                         out.println("false");
13                 out.flush();
14         }else if( task.equals("remove")){
15                 removePlayer(login);
16         }else if( task.equals("joinmatch")){
17                 String loginHost = datas[3];
18                 if( ((Player) liste.get(loginHost)) == null || ((Player)
                       liste.get(login)) == null ){
19                         out.println("false");
20                 }else{
21                         Match m = ((Player) liste.get(loginHost)).getMatch()
                               ;
22                         if( m == null )
23                                 out.println("false");
24                         else{
25                                 ((Player) liste.get(login)).setMatch(m);
26                                 out.println(m.addPlayer(((Player) liste.get(
                                       login))));
27                         }
28                 }
29                 out.flush();
30         }
31         }
32
33         private void removePlayer(String login) {
34                 if( liste.containsKey(login) && liste.get(login) != null){
35                         if( ((Player)liste.get(login)).getMatch() != null )
36                         ((Player)liste.get(login)).getMatch().removePlayer(
                               login);
37                         ServerBabyfoot.tmatch.removeFromListe(((Player)liste
                               .get(login)).getMatch());
38                         liste.remove(login);
39                 }
40         }
41
42         private boolean addPlayer(String login) {
43                 if( !liste.containsKey(login)){
44                         liste.put(login,new Player(login));
45                         return true;
46                 }
47                 return false;
48         }
49
50         public Player getPlayer(String login) {
```

```
51                  if( liste.containsKey(login)){
52                          return liste.get(login);
53                  }
54                  return null;
55          }
56  }
```

2.2.2 ServerBabyfoot.java

```
1   public class ServerBabyfoot implements Runnable {
2
3           ServerSocket socketserver = null;
4           Socket socket = null;
5           BufferedReader in;
6           PrintWriter out;
7           String login;
8           public static ChatServer tchat;
9           public static PlayerServer tplayer;
10          public static MatchServer tmatch;
11
12          public ServerBabyfoot(ServerSocket s){
13                  tmatch = new MatchServer();
14                  tplayer = new PlayerServer();
15                  tchat = new ChatServer();
16                  socketserver = s;
17          }
18
19
20          public void run() {
21                  try {
22              while(true){
23                  socket = socketserver.accept();
24                  in = new BufferedReader(new InputStreamReader(socket.
                        getInputStream()));
25                          out = new PrintWriter(socket.getOutputStream());
26                          Thread allocator = new Thread(new Allocator(in, out)
                            );
27                          allocator.start();
28              }
29          } catch (IOException e) {
30              System.err.println("Erreur serveur à la réception !");
31          }
32          }
33
34          public static void main(String[] args){
35                  int port = 2010;
36                  if( args.length >= 1 ){
37                          if( args[0].equals("-port") && args[1].length() > 0
                            ){
38                                  port = Integer.valueOf(args[1]);
39                          }
40                  }
41                  try{
42                          System.out.println("Lancement du serveur en cours...
                            ");
43                          ServerBabyfoot s = new ServerBabyfoot(new
                            ServerSocket(port));
44                          System.out.println("Serveur prêt !");
45                          Thread serveur = new Thread(s);
```

```
46                          serveur.start();
47                  } catch (IOException e) {
48                  System.err.println("Erreur serveur au lancement !");
49              }
50          }
51  }
52
53
54  class Allocator implements Runnable{
55          BufferedReader in;
56          PrintWriter out;
57          public Allocator( BufferedReader in, PrintWriter out ){
58                  this.in = in;
59                  this.out = out;
60          }
61
62          public void run(){
63                  String m;
64                  try {
65                          while(true){
66                                  m = in.readLine();
67                                  if( m != null ) {
68                                          String[] datas = m.split(Utils.
                                             SEPARATOR);
69                                          String typeRequete = datas[0];
70                                          if( typeRequete.equals("player") ){
71                                                  ServerBabyfoot.tplayer.
                                                     setQuery(m);
72                                                  ServerBabyfoot.tplayer.
                                                     handle(in, out);
73                                          }else if( typeRequete.equals("match"
                                             ) ){
74                                                  ServerBabyfoot.tmatch.
                                                     setQuery(m);
75                                                  ServerBabyfoot.tmatch.handle
                                                     (in, out);
76                                          }else if( typeRequete.equals("
                                             servers") ){
77                                                  ServerBabyfoot.tchat.
                                                     setQuery(m);
78                                                  ServerBabyfoot.tchat.handle(
                                                     in, out);
79                                          }else if( typeRequete.equals("chat")
                                              ){
80                                                  ServerBabyfoot.tchat.
                                                     setQuery(m);
81                                                  ServerBabyfoot.tchat.handle(
                                                     in, out);
82                                          }
83                          }else{
84                                  break;
85                          }
86                          }
87                  } catch (IOException e) {
88                          e.printStackTrace();
89                  }
90          }
91  }
```