

Rapport du projet informatique

Matthieu Denoux - Groupe 1

25 mars 2014

Table des matières

I	Analyse du travail effectu[Pleaseinsertintopreamble]	3
1	Présentation du sujet	4
1.1	Principe général	4
1.2	L'interface graphique	4
2	Analyse de la solution envisagée	6
2.1	Découpage en modules	6
2.2	L'interface graphique	6
2.3	Le réseau	7
2.3.1	Le principe général	7
2.3.2	La partie client	7
2.3.3	La partie serveur	8
2.4	Le cœur algorithmique	9
3	Échéancier	11
II	Explications des choix techniques	12
3.1	Le réseau	13
3.2	La gestion du jeu	13
III	Le code partiel du programme	14
1	Client	15
1.1	Core	15
1.1.1	ClientBabyfoot.java	15
1.1.2	Utils.java	16
1.2	Gui	18
1.2.1	BPanel.java	18
1.2.2	GameZone.java	19
1.3	Network	24
1.3.1	Client.java	24
1.3.2	MatchClient.java	25
1.4	Classe de test	31
2	Serveur	33
2.1	Core	33
2.1.1	Fonction de tir dans Match.java	33
2.1.2	Thread de Match.java	36
2.1.3	Collisions.java	38

2.2	Network	44
2.2.1	PlayerServer.java	44
2.2.2	ServerBabyfoot.java	45

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 incluse 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 applquette.

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*.

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*.

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.

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 déplacements. Toutes les informations, les calculs sont effectués par le serveur et envoyées.

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.

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 Chat-Panel, 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-Utills.LINE_STRENGTH-  
Utills.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** : déboguer 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.

3.2 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 légèrement 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){
```



```

36         e.printStackTrace();
37     }
38 }
39 System.exit(0);
40 }
41
42 public void init(){
43     setClient(new Client(this));
44     setChat(new Chat(this));
45     setPlayer(new Player(this));
46     mainFrame = new MainFrame("Babyfoot en réseau trololol",
47         this);
48 }

```

1.1.2 Utils.java

```

1  /** Cette classe dispose de fonctions utiles qui peuvent être nécessaires par
2   toutes les parties du programme. Elle contient aussi
3   les constantes nécessaires (par exemple de type enum) concernant le
4   programme. */
5  public abstract class Utils {
6      public static enum Types { ONEVSTONE, TWOVSTWO, ONEVSTWO };
7      public static enum States { WAITING, FULL, PLAYING, FINISHED };
8      public static enum Sides { DOWN, UP };
9
10     public static final int MATCH_END = 100;
11
12     public static enum Rod { GARDIEN , DEFENSE, MILIEU, ATTAQUE };
13     public static enum RodStatus { NORMAL, SHOOTING, HOLDING };
14
15     public static enum CollisionType { SIDES, UPANDDDOWN };
16
17     public static final String SEPARATOR = ";";
18
19     public static final int GOAL_SIZE = 2*100;
20     public static final int LINE_STRENGTH = 4;
21     public static final int GAP_EDGE = 2*20;
22     public static final int IMAGE_PLAYER_Y = 38;
23     public static final int IMAGE_PLAYER_X = 30;
24     public static final int MOVE_STEP = 10;
25     public static final int BALL_RADIUS = 15;
26
27     public static final int HEIGHT = 700;
28     public static final int WIDTH = 900;
29
30     public static final int MAX_INITIAL_SPEED = 4;
31
32     public static final int GARDIEN_POSITION = GAP_EDGE+30;
33     public static final int DEFENSE_POSITION = GAP_EDGE+30+100;
34     public static final int MILIEU_POSITION = (WIDTH-LINE_STRENGTH)
35         /2-70;
36     public static final int ATTAQUE_POSITION = WIDTH-Utils.LINE_STRENGTH
37         -Utils.GAP_EDGE-230;
38
39     private static int sensibility = 30;
40
41     @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

```

```

2 | 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
3 | du BorderLayout. */
4 | public abstract class BPanel extends JPanel {
5 |     private MainFrame window;
6 |
7 |     public BPanel(MainFrame f, boolean display, int heightHeader ){
8 |         setWindow(f);
9 |         getWindow().getMain().actualPanel = this;
10 |        setLayout(new BorderLayout());
11 |        setBackground(Color.WHITE);
12 |        add(new Header(display, heightHeader),BorderLayout.NORTH);
13 |    }
14 |
15 |    public BPanel(MainFrame f ){
16 |        this( f, true, 150 );
17 |    }
18 |    public MainFrame getWindow() {
19 |        return window;
20 |    }
21 |    public void setWindow(MainFrame window) {
22 |        this.window = window;
23 |    }
24 | }

```

1.2.2 GameZone.java

```

1 | public class GameZone extends JPanel implements KeyListener,
   |     MouseMotionListener, MouseListener {
2 |     private static final long serialVersionUID = 1L;
3 |     /**
4 |      * Taille des buts en pixels et épaisseur des traits
5 |      */
6 |     private int oldY = 0;
7 |
8 |     private Hashtable<Rod, Integer>[] yPosition;
9 |
10 |    private Hashtable<Rod, Boolean> rodPositions;
11 |
12 |    private Hashtable<Rod, RodStatus>[] rodStatus;
13 |
14 |    Rod rodPosition;
15 |
16 |    private int ballX;
17 |    private int ballY;
18 |
19 |
20 |    private GamePanel gamepanel;
21 |    private InfoZone infoZone;
22 |
23 |    private long shootBeginning;
24 |    private boolean pause = false;
25 |    private Sides side;
26 |    private int lastKeyY;
27 |    private Sides toNormalSide;
28 |    private Rod toNormalRod;
29 |
30 |    @SuppressWarnings("unchecked")
31 |    public GameZone(GamePanel window, boolean testMode, int width){

```

```

32         this.gamepanel = window;
33
34         this.setSide(getGamePanel().getWindow().getPlayer().getSide
35             ());
36         setSize(width,729);
37
38         infoZone = new InfoZone(this);
39         setRodStatus(new Hashtable[2]);
40         rodStatus[0] = new Hashtable<Rod, RodStatus>();
41         rodStatus[0].put(Rod.GARDIEN, RodStatus.NORMAL);
42         rodStatus[0].put(Rod.DEFENSE, RodStatus.NORMAL);
43         rodStatus[0].put(Rod.MILIEU, RodStatus.NORMAL);
44         rodStatus[0].put(Rod.ATTAQUE, RodStatus.NORMAL);
45         rodStatus[1] = new Hashtable<Rod, RodStatus>();
46         rodStatus[1].put(Rod.GARDIEN, RodStatus.NORMAL);
47         rodStatus[1].put(Rod.DEFENSE, RodStatus.NORMAL);
48         rodStatus[1].put(Rod.MILIEU, RodStatus.NORMAL);
49         rodStatus[1].put(Rod.ATTAQUE, RodStatus.NORMAL);
50
51         yPosition = new Hashtable[2];
52         yPosition[0] = new Hashtable<Rod, Integer>();
53         yPosition[0].put(Rod.GARDIEN, Utils.Y_STAGGERING_DEFAULT.get
54             (Rod.GARDIEN));
55         yPosition[0].put(Rod.DEFENSE, Utils.Y_STAGGERING_DEFAULT.get
56             (Rod.DEFENSE));
57         yPosition[0].put(Rod.MILIEU, Utils.Y_STAGGERING_DEFAULT.get(
58             Rod.MILIEU));
59         yPosition[0].put(Rod.ATTAQUE, Utils.Y_STAGGERING_DEFAULT.get
60             (Rod.ATTAQUE));
61         yPosition[1] = new Hashtable<Rod, Integer>();
62         yPosition[1].put(Rod.GARDIEN, Utils.Y_STAGGERING_DEFAULT.get
63             (Rod.GARDIEN));
64         yPosition[1].put(Rod.DEFENSE, Utils.Y_STAGGERING_DEFAULT.get
65             (Rod.DEFENSE));
66         yPosition[1].put(Rod.MILIEU, Utils.Y_STAGGERING_DEFAULT.get(
67             Rod.MILIEU));
68         yPosition[1].put(Rod.ATTAQUE, Utils.Y_STAGGERING_DEFAULT.get
69             (Rod.ATTAQUE));
70
71         rodPosition = ( getGamePanel().getWindow().getMain().
72             getPlayer().getRodAvailables().get(Rod.MILIEU) ? Rod.
73             MILIEU : Rod.GARDIEN );
74         setPreferredSize(new Dimension(width,729));
75         setMinimumSize(new Dimension(width,729));
76
77         addKeyListener(this);
78         addMouseListener(this);
79         addMouseMotionListener(this);
80
81         setFocusable(true);
82         requestFocusInWindow();
83
84         if( !testMode ){
85             Thread tr = new Thread(new RefreshRods(this));
86             tr.start();
87         }
88     }
89
90     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                 );
87             } catch (IOException e) {
88                 e.printStackTrace();
89             }
90
91             initInfoZone();
92             drawLines(g);
93             drawBall(g);
94             drawGoals(g);
95             drawPlayers(g);
96             drawRodPosition(g);
97         }
98
99         private void initInfoZone(){
100             infoZone.getLeftScore().setBackground(Color.RED);
101             infoZone.getRightScore().setBackground(Color.BLUE);
102             gamepanel.repaint();
103         }
104
105         private void drawBall(Graphics g) {
106             //Côté gauche
107             g.setColor(Color.WHITE);
108             g.fillOval(ballX-Utills.BALL_RADIUS, ballY-Utills.BALL_RADIUS,
109                 Utills.BALL_RADIUS*2, Utills.BALL_RADIUS*2);
110         }
111
112         private void drawGoals(Graphics g){
113             //Côté gauche
114             g.setColor(Color.RED);
115             //Long côté
116             g.fillRect(Utills.GAP_EDGE/2,Utills.HEIGHT/2-Utills.GOAL_SIZE
117                 /2,Utills.LINE_STRENGTH,Utills.GOAL_SIZE);
118             //petits côtés
119             g.fillRect(Utills.GAP_EDGE/2,Utills.HEIGHT/2-Utills.GOAL_SIZE
120                 /2,Utills.GAP_EDGE/2,Utills.LINE_STRENGTH);
121             g.fillRect(Utills.GAP_EDGE/2,Utills.HEIGHT/2+Utills.GOAL_SIZE
122                 /2,Utills.GAP_EDGE/2,Utills.LINE_STRENGTH);
123             //Côté droit
124             g.setColor(Color.BLUE);
125             //Long côté
126             g.fillRect(Utills.WIDTH-Utills.GAP_EDGE/2-Utills.LINE_STRENGTH,
127                 Utills.HEIGHT/2-Utills.GOAL_SIZE/2,Utills.LINE_STRENGTH,
128                 Utills.GOAL_SIZE);
129             //petits côtés
130             g.fillRect(Utills.WIDTH-Utills.GAP_EDGE,Utills.HEIGHT/2-Utills.
131                 GOAL_SIZE/2,Utills.GAP_EDGE/2,Utills.LINE_STRENGTH);
132             g.fillRect(Utills.WIDTH-Utills.GAP_EDGE,Utills.HEIGHT/2+Utills.
133                 GOAL_SIZE/2,Utills.GAP_EDGE/2,Utills.LINE_STRENGTH);
134         }
135
136         private void drawLines(Graphics g){
137             try {
138                 Image img = ImageIO.read(new File("pictures/bordhaut.png"));
139                 g.drawImage(img, 0, 0, this);
140             }
141         }

```

```

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 ){

```

```

173         RodStatus status = rodStatus[side == Sides.UP ? 1 : 0].get(
174             rod);
175         int position = status == RodStatus.HOLDING ? 3 : (status ==
176             RodStatus.NORMAL ? 1 : 2);
177         g.setColor(new Color(192, 192, 192));
178         g.fillRect(x, 20, 3*Utils.LINE_STRENGTH/2, Utils.HEIGHT-92+52);
179         g.setColor(color);
180         for( int i=1; i<=nb; i++){
181             try {
182                 Image img = null;
183                 if(side == Sides.UP)
184                     img = ImageIO.read(new File("
185                         pictures/joueurdroit" + String.
186                             valueOf(position) + ".png"));
187                 else
188                     img = ImageIO.read(new File("
189                         pictures/joueurgauche" + String.
190                             valueOf(position) + ".png"));
191                 if(position==1)
192                     g.drawImage(img, x-Utils.
193                         IMAGE_PLAYER_X/3, Utils.
194                             getYPositionPlayer(yPosition, rod
195                                 , i, nb, side), this);
196                 else if(position==2 && side == Sides.DOWN )
197                     g.drawImage(img, x-Utils.
198                         IMAGE_PLAYER_X/3-30, Utils.
199                             getYPositionPlayer(yPosition, rod
200                                 , i, nb, side), this);
201                 else if(position==2 && side == Sides.UP )
202                     g.drawImage(img, x-Utils.
203                         IMAGE_PLAYER_X/3-60, Utils.
204                             getYPositionPlayer(yPosition, rod
205                                 , i, nb, side), this);
206                 else if(position==3 && side == Sides.DOWN )
207                     g.drawImage(img, x-Utils.
208                         IMAGE_PLAYER_X/3-40, Utils.
209                             getYPositionPlayer(yPosition, rod
210                                 , i, nb, side), this);
211                 else if(position==3 && side == Sides.UP )
212                     g.drawImage(img, x-Utils.
213                         IMAGE_PLAYER_X/3-40, Utils.
214                             getYPositionPlayer(yPosition, rod
215                                 , i, nb, side), this);
216             } catch (IOException e) {
217                 e.printStackTrace();
218             }
219         }
220     }

```

```

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.getBallyY() );
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
23                                 le chat");
24             cc = new ChatClient( Client.socketChat, main );
25             tChat = new Thread(getCc());
26             tChat.start();
27             Client.socketPlayer = new Socket("127.0.0.1",main.getPort())
28             ;
29             System.out.println("Connexion établie avec le serveur pour
30                                 les joueurs");
31             pc = new PlayerClient( Client.socketPlayer, main );
32             tPlayer = new Thread(pc);
33             tPlayer.start();
34             Client.socketMatch = new Socket("127.0.0.1",main.getPort());
35             System.out.println("Connexion établie avec le serveur pour
36                                 les matchs");
37             mc = new MatchClient( Client.socketMatch );
38             tMatch = new Thread(mc);
39             tMatch.start();
40             Client.socketGame = new Socket("127.0.0.1",main.getPort());
41             System.out.println("Connexion établie avec le serveur pour
42                                 les jeux");
43             gc = new GameClient( Client.socketGame, main );
44             tGame = new Thread(gc);
45             tGame.start();
46         } catch (UnknownHostException e) {
47             System.err.println("Impossible de se connecter à l'adresse
48                                 127.0.0.1 ");
49         } catch (IOException e) {
50             System.err.println("Aucun serveur à l'écoute du port " + main.
51                                 getPort());
52         }
53         sc.close();
54     }

```

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.
27             getInputStream()));
28         prc = new MatchReceptionMessage(in, this);
29         Thread tReceptionMessage = new Thread(prc);
30         tReceptionMessage.start();
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,
37     RodStatus[] rodStatus) {
38     out.println("match" + Utils.SEPARATOR + "setrod" + Utils.
39         SEPARATOR + login + Utils.SEPARATOR
40         + rodPositions[0] + Utils.SEPARATOR +
41         rodPositions[1] + Utils.SEPARATOR +
42         rodPositions[2] + Utils.SEPARATOR +
43         rodPositions[3]
44         + Utils.SEPARATOR + rodStatus[0] + Utils.
45         SEPARATOR + rodStatus[1] + Utils.
46         SEPARATOR + rodStatus[2] + Utils.
47         SEPARATOR + rodStatus[3]);
48
49     out.flush();
50 }
51
52 public String[] getServers() {
53     out.println("match" + Utils.SEPARATOR + "getserverslist" );
54     out.flush();
55     try {
56         Thread.currentThread();
57         Thread.sleep(100);
58     } catch (InterruptedException e) {
59         e.printStackTrace();
60     }
61     return getServerList();
62 }
63
64 public boolean setServerFromHost(String login, String loginHost) {
65     out.println("player" + Utils.SEPARATOR + "joinmatch" + Utils.
66         SEPARATOR + login + Utils.SEPARATOR + loginHost );
67     out.flush();
68     try {
69         Thread.currentThread();
70         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" +
74             Utils.SEPARATOR + login );
75         out.flush();
76         try {
77             Thread.currentThread();
78             Thread.sleep(200);
79         } catch (InterruptedException e) {
80             e.printStackTrace();
81         }
82         return getMatchDatas();
83     }
84
85     public String[] getServerList() {
86         return serverList;
87     }
88
89     public void setServerList(String[] serverList) {
90         this.serverList = serverList;
91     }
92
93     public boolean isOk() {
94         return ok;
95     }
96
97     public void setOk(boolean ok) {
98         this.ok = ok;
99     }
100
101     public void setMatchDatas(String[] md) {
102         switch( Integer.valueOf( md[1] ) ){
103             case 1:
104                 matchDatas = new String[4];
105                 matchDatas[1] = "1";
106                 if( md[2].equals(" ") ){
107                     matchDatas[2] = "";
108                     matchDatas[3] = md[3];
109                 }else if( md[3].equals(" ") ){
110                     matchDatas[2] = md[2];
111                     matchDatas[3] = md[4];
112                 }
113                 break;
114             case 2:
115                 matchDatas = new String[6];
116                 matchDatas[1] = "2";
117                 if( md[2].equals(" ") ){
118                     matchDatas[2] = "";
119                     matchDatas[3] = "";
120                     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.
170         SEPARATOR + login );
171     out.flush();
172     try {
173         Thread.currentThread();
174         Thread.sleep(200);
175     } catch (InterruptedException e) {
176         e.printStackTrace();
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     BufferedReader in;
276     boolean ready;
277     MatchClient matchClient;
278
279     public MatchReceptionMessage(BufferedReader in, MatchClient
280         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         );
5         Thread serveur = new Thread(s);
6         serveur.start();
7
8         ClientBabyfoot m1 = new ClientBabyfoot(2010);
9         Thread t1 = new Thread(new TesterThread(m1));
10        t1.start();
11    }

```



```

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(
23             new NewPanel(((MenuPanel) main.actualPanel).getWindow()))
24             ;
25         ((NewPanel) main.actualPanel).getWindow().setVisible(true);
26         ((NewPanel) main.actualPanel).getWindow().getMain().
27             getPlayer().addMatch(1);
28         ((NewPanel) main.actualPanel).getWindow().getMain().
29             getPlayer().setBoss(true);
30         ((NewPanel) main.actualPanel).getWindow().setContentPane(new
31             WaitingRoomPanel(((NewPanel) main.actualPanel).getWindow
32             ()));
33         ((WaitingRoomPanel) main.actualPanel).getWindow().setVisible
34             (true);
35         PrintWriter out = ((WaitingRoomPanel) main.actualPanel).
36             getWindow().getMain().getClient().getMc().getOut();
37         String login = "coucou";
38         out.println("player" + Utils.SEPARATOR + "add" + Utils.
39             SEPARATOR + login);
40         out.flush();
41         out.println("player" + Utils.SEPARATOR + "joinmatch" + Utils
42             .SEPARATOR + login + Utils.SEPARATOR + loginHost );
43         out.flush();
44         out.println("match" + Utils.SEPARATOR + "run" + Utils.
45             SEPARATOR + loginHost );
46         out.flush();
47         ((WaitingRoomPanel) main.actualPanel).getWindow().getMain().
48             getClient().getGc().startThread();
49         ((WaitingRoomPanel) main.actualPanel).getWindow().
50             setContentPane(new GamePanel(((WaitingRoomPanel) main.
51             actualPanel).getWindow(),false));
52         ((WaitingRoomPanel) main.actualPanel).getWindow().
53             getContentPane().requestFocusInWindow();
54         ((WaitingRoomPanel) main.actualPanel).getWindow().setVisible
55             (true);
56     }
57 }

```

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(
7                      yRodPositions, Rod.GARDIEN, 1, 1, side );
8                  if( ballX >= Utils.GARDIEN_POSITION && ballX
9                      <= Utils.GARDIEN_POSITION + Utils.
10                         IMAGE_PLAYER_SHOOTING_X + Utils.
11                         BALL_RADIUS
12                         //&& ballSpeedX <= 0
13                         && ballY >= yPosition &&
14                         ballY <= Utils.
15                             IMAGE_PLAYER_Y +
16                             yPosition ){
17                     ballSpeedX = Utils.MAX_INITIAL_SPEED
18                     ;
19                     ballSpeedY *= -0.1;
20                 }
21             }else if( rod == Rod.DEFENSE ){
22                 boolean test = false;
23                 for( int i = 1; i <= 2; i++ ){
24                     int yPosition = Utils.
25                         getYPositionPlayer(yRodPositions,
26                             Rod.DEFENSE, i, 2, side );
27                     test |= ( ballY >= yPosition &&
28                             ballY <= Utils.IMAGE_PLAYER_Y +
29                             yPosition );
30                     if( test ) break;
31                 }
32                 if( ballX >= Utils.DEFENSE_POSITION && ballX
33                     <= Utils.DEFENSE_POSITION + Utils.
34                         IMAGE_PLAYER_SHOOTING_X + Utils.
35                         BALL_RADIUS
36                         //&& ballSpeedX <= 0
37                         && test ){
38                     ballSpeedX = Utils.MAX_INITIAL_SPEED
39                     ;
40                     ballSpeedY *= -0.1;
41                 }
42             }
43         }
44     }
```

```

25     }
26 }else if( rod == Rod.MILIEU ){
27     boolean test = false;
28     for( int i = 1; i <= 5; i++ ){
29         int yPosition = Utils.
30             getYPositionPlayer(yRodPositions,
31                 Rod.MILIEU, i, 5, side );
32         test |= ( ballyY >= yPosition &&
33             ballyY <= Utils.IMAGE_PLAYER_Y +
34                 yPosition );
35         if( test ) break;
36     }
37     if( ballX >= Utils.MILIEU_POSITION && ballX
38         <= Utils.MILIEU_POSITION + Utils.
39             IMAGE_PLAYER_SHOOTING_X + Utils.
40                 BALL_RADIUS
41             //&& ballSpeedX <= 0
42             && test ){
43         ballSpeedX = Utils.MAX_INITIAL_SPEED
44             ;
45         ballSpeedY *= -0.1;
46     }
47 }else if( rod == Rod.ATTAQUE ){
48     boolean test = false;
49     for( int i = 1; i <= 3; i++ ){
50         int yPosition = Utils.
51             getYPositionPlayer(yRodPositions,
52                 Rod.ATTAQUE, i, 3, side );
53         test |= ( ballyY >= yPosition - Utils.
54             .BALL_RADIUS && ballyY <= Utils.
55                 BALL_RADIUS + Utils.
56                 IMAGE_PLAYER_Y + yPosition );
57         if( test ) break;
58     }
59     if( ballX >= Utils.ATTAQUE_POSITION && ballX
60         <= Utils.ATTAQUE_POSITION + Utils.
61             IMAGE_PLAYER_SHOOTING_X + Utils.
62                 BALL_RADIUS
63             //&& ballSpeedX <= 0
64             && test ){
65         ballSpeedX = Utils.MAX_INITIAL_SPEED
66             ;
67         ballSpeedY *= -0.1;
68     }
69 }
70 }else{
71     if( rod == Rod.GARDIEN ){
72         int yPosition = Utils.getYPositionPlayer(
73             yRodPositions, Rod.GARDIEN, 1, 1, side);
74         if( ballX + Utils.IMAGE_PLAYER_SHOOTING_X +
75             Utils.BALL_RADIUS >= Utils.
76                 GARDIEN_POSITION && ballX <= Utils.
77                     GARDIEN_POSITION
78             //&& ballSpeedX <= 0
79             && ballyY >= yPosition &&
80                 ballyY <= Utils.
81                     IMAGE_PLAYER_Y +
82                         yPosition ){
83             ballSpeedX = Utils.MAX_INITIAL_SPEED
84                 ;

```

```

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.
66                 getYPositionPlayer(yRodPositions,
67                     Rod.DEFENSE, i, 2, side);
68             test |= ( ballY >= yPosition &&
69                 ballY <= Utils.IMAGE_PLAYER_Y +
70                 yPosition );
71             if( test ) break;
72         }
73         if( ballX + Utils.IMAGE_PLAYER_SHOOTING_X +
74             Utils.BALL_RADIUS >= Utils.
75             DEFENSE_POSITION && ballX <= Utils.
76             DEFENSE_POSITION
77             //&& ballSpeedX <= 0
78             && test ){
79             ballSpeedX = Utils.MAX_INITIAL_SPEED
80                 ;
81             ballSpeedY *= -0.1;
82         }
83     }else if( rod == Rod.MILIEU ){
84         boolean test = false;
85         for( int i = 1; i <= 5; i++ ){
86             int yPosition = Utils.
87                 getYPositionPlayer(yRodPositions,
88                     Rod.MILIEU, i, 5, side);
89             test |= ( ballY >= yPosition &&
90                 ballY <= Utils.IMAGE_PLAYER_Y +
91                 yPosition );
92             if( test ) break;
93         }
94         if( ballX + Utils.IMAGE_PLAYER_SHOOTING_X +
95             Utils.BALL_RADIUS >= Utils.
96             MILIEU_POSITION && ballX <= Utils.
97             MILIEU_POSITION
98             //&& ballSpeedX <= 0
99             && test ){
100             ballSpeedX = Utils.MAX_INITIAL_SPEED
101                 ;
102             ballSpeedY *= -0.1;
103         }
104     }else if( rod == Rod.ATTAQUE ){
105         boolean test = false;
106         for( int i = 1; i <= 3; i++ ){
107             int yPosition = Utils.
108                 getYPositionPlayer(yRodPositions,
109                     Rod.ATTAQUE, i, 3, side);
110             test |= ( ballY >= yPosition - Utils.
111                 BALL_RADIUS && ballY <= Utils.
112                 BALL_RADIUS + Utils.
113                 IMAGE_PLAYER_Y + yPosition );
114             if( test ) break;
115         }
116         if( ballX + Utils.IMAGE_PLAYER_SHOOTING_X +
117             Utils.BALL_RADIUS >= Utils.
118             ATTAQUE_POSITION && ballX <= Utils.
119             ATTAQUE_POSITION

```

```

96         // && ballSpeedX <= 0
97         && test ){
98             ballSpeedX = Utils.MAX_INITIAL_SPEED
99             ;
100             ballSpeedY *= -0.1;
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
15                     // gauche et que la vitesse est bien
16                     // négative (donc vers la gauche),
17                     // on change de vitesse.
18                     if( ( match.getBallX() -5 - Utils.
19                         BALL_RADIUS ) <= ( Utils.
20                         GAP_EDGE + Utils.LINE_STRENGTH )
21                     ) {
22                         if( match.getBallSpeedX() <
23                             0 )
24                             match.setBallSpeedX
25                                 ((-1)*match.
26                                     getBallSpeedX()+
27                                     match.isSlow() ?
28                                     0 : match.
29                                     getBallSpeedX()/
30                                     Math.abs(match.
31                                     getBallSpeedX()))
32                                 );
33                         match.verifGoal();
34                     // Si on a atteint le bord extérieur
35                     // droit et que la vitesse est bien
36                     // positive (donc vers la droite),
37                     // on change de vitesse
38                     }else if( ( match.getBallX()+5 ) >=
39                         ( Utils.WIDTH - Utils.GAP_EDGE -
40                         Utils.LINE_STRENGTH - Utils.
41                         BALL_RADIUS ) ){
42                         if( match.getBallSpeedX() >
43                             0 )
44                             match.setBallSpeedX
45                                 ((-1)*match.

```

```

23         getBallSpeedX()+(
24             match.isSlow() ?
25             0 : match.
26             getBallSpeedX()/
27             Math.abs(match.
28                 getBallSpeedX()))
29             );
30         match.verifGoal();
31     }
32     //Si on a atteint le bord extérieur
33     //haut et que la vitesse est bien n
34     //égative (donc vers le haut), on
35     //change de vitesse.
36     if( ( match.getBallY()-5 - Utils.
37         BALL_RADIUS ) <= ( Utils.
38         GAP_EDGE + Utils.LINE_STRENGTH )
39     ){
40         if( match.getBallSpeedY() <
41             0 )
42             match.setBallSpeedY
43                 ((-1)*match.
44                 getBallSpeedY()+(
45                 match.isSlow() ?
46                 0 : match.
47                 getBallSpeedY()/
48                 Math.abs(match.
49                     getBallSpeedY()))
50             );
51         //Si on a atteint le bord extérieur
52         //bas et que la vitesse est bien
53         //positive (donc vers le bas), on
54         //change de vitesse
55     }else if( ( match.getBallY() +5+
56         Utils.BALL_RADIUS ) >= ( Utils.
57         HEIGHT - Utils.GAP_EDGE - Utils.
58         LINE_STRENGTH ) ) {
59         if( match.getBallSpeedY() >
60             0 )
61             match.setBallSpeedY
62                 ((-1)*match.
63                 getBallSpeedY()+(
64                 match.isSlow() ?
65                 0 : match.
66                 getBallSpeedY()/
67                 Math.abs(match.
68                     getBallSpeedY()))
69             );
70     }
71
72     CollisionType resultatsCollisions =
73         match.testCollisions();
74     if( resultatsCollisions ==
75         CollisionType.SIDES )
76         match.setBallSpeedX((-1)*
77             match.getBallSpeedX()+(
78             match.isSlow() ? 0 : 1*
79             match.getBallSpeedX()/11)
80         );
81     else if( resultatsCollisions ==
82         CollisionType.UPANDDOWN )

```

```

39         match.setBallSpeedY((-1)*
40             match.getBallSpeedY()+
41             match.isSlow() ? 0 : 1*
42             match.getBallSpeedY()/11
43             );
44     }
45 }
46 }catch(InterruptedExecution 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            ;
16        Hashtable<Rod, Long> tUp = new Hashtable<Rod, Long>();
17        Hashtable<Rod, Long> tDown = new Hashtable<Rod, Long>();
18
19        lastCollision.put(Sides.UP, tUp);
20        lastCollision.put(Sides.DOWN, tDown);
21        zeroLast();
22    }
23
24    private void zeroLast(){
25        lastCollision.get(Sides.UP).put(Rod.GARDIEN, 0L);
26        lastCollision.get(Sides.UP).put(Rod.DEFENSE, 0L);
27        lastCollision.get(Sides.UP).put(Rod.MILIEU, 0L);
28        lastCollision.get(Sides.UP).put(Rod.ATTAQUE, 0L);
29        lastCollision.get(Sides.DOWN).put(Rod.GARDIEN, 0L);
30        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.
41                 DOWN );
42             Sides otherSide = ( rodTop != null ? Sides.DOWN :
43                 Sides.UP );
44             if( (lastCollision.get(side)).get(rod) > 0L &&
45                 System.currentTimeMillis() - (lastCollision.get(
46                     side)).get(rod) < 500 ){
47                 if( (lastCollision.get(otherSide)).get(rod)
48                     > System.currentTimeMillis() - 500 ) {
49                     (lastCollision.get(otherSide)).put(
50                         rod, System.currentTimeMillis());
51                 }else{
52                     //System.out.println("Avortée !" +
53                         side + " - " + rod);
54                     return null;
55                 }
56             }
57             //System.out.println("Avortée - " + side + "
58                 - " + rod);
59             return null;
60         }
61         else{
62             //zeroLast();
63             (lastCollision.get(side)).put(rod, System.
64                 currentTimeMillis());
65         }
66     }
67
68     public CollisionType testCollisionsTop(Integer position, Rod rod){
69         //y final : y + i*h/(1+nb)-Utils.IMAGE_PLAYER_Y/2 + yDecal[
70             rightPlayer ? 1 : 0].get(rod)-Utils.Y_STAGGERING_DEFAULT.
71             get(rod)
72         // i va de 1 à nb où nb est le nombre de joueurs sur une
73             barre
74         float yTopHitBox = position + Utils.GAP_EDGE;
75         float xLeftHitBox = 0;
76         CollisionType ballPosition = null;
77         switch(rod){
78             //Utils.WIDTH-Utils.LINE_STRENGTH-Utils.GARDIEN_POSITION-
79                 Utils.IMAGE_PLAYER_X/3
80             case GARDIEN:
81                 //yTopHitBox = position + Utils.HEIGHT/2-
82                     Utils.IMAGE_PLAYER_Y/2 - Utils.
83                     Y_STAGGERING_DEFAULT.get(Rod.GARDIEN);
84                 yTopHitBox = Utils.getYPositionPlayer(
85                     position, rod, 1, 1 );
86                 xLeftHitBox = Utils.WIDTH - (Utils.
87                     GARDIEN_POSITION+(float)(Utils.
88                     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(
134             position, rod, 2, 2 );
135         ballPosition = isBallInCollision(
136             xLeftHitBox, yTopHitBox, xLeftHitBox +
137             Utils.IMAGE_PLAYER_X, yTopHitBox + Utils.
138             IMAGE_PLAYER_Y );
139         if( ballPosition != null )
140             return ballPosition;
141         break;
142     case MILIEU:
143         xLeftHitBox = Utils.MILIEU_POSITION-(float)(
144             Utils.IMAGE_PLAYER_X/3);
145         for( int i = 1; i < 6; i++ ){
146             //yTopHitBox = position + i*Utils.
147             //HEIGHT/6-Utills.IMAGE_PLAYER_Y/2 -
148             //Utils.Y_STAGGERING_DEFAULT.get(
149             //Rod.MILIEU);
150             yTopHitBox = Utils.
151             getYPositionPlayer( position, rod
152             , i, 5 );
153             ballPosition = isBallInCollision(
154                 xLeftHitBox, yTopHitBox,
155                 xLeftHitBox + Utils.
156                 IMAGE_PLAYER_X, yTopHitBox +
157                 Utils.IMAGE_PLAYER_Y );
158             if( ballPosition != null )
159                 return ballPosition;
160         }
161         break;
162     case ATTAQUE:
163         xLeftHitBox = Utils.ATTAQUE_POSITION-(float)
164         (Utils.IMAGE_PLAYER_X/3);
165         for( int i = 1; i < 4; i++ ){
166             //yTopHitBox = position + i*Utils.
167             //HEIGHT/4-Utills.IMAGE_PLAYER_Y/2 -
168             //Utils.Y_STAGGERING_DEFAULT.get(
169             //Rod.ATTAQUE);
170             yTopHitBox = Utils.
171             getYPositionPlayer( position, rod
172             , i, 3 );
173             ballPosition = isBallInCollision(
174                 xLeftHitBox, yTopHitBox,
175                 xLeftHitBox + Utils.
176                 IMAGE_PLAYER_X, yTopHitBox +
177                 Utils.IMAGE_PLAYER_Y );
178             if( ballPosition != null )
179                 return ballPosition;
180         }
181         break;
182     }
183     return null;
184 }
185
186 public CollisionType isBallInCollision(float xLeftTop, float
187 yLeftTop, float xRightBottom, float yRightBottom ){
188     /*if( ( Math.pow( ballX - xLeftTop, 2) + Math.pow( ballY -
189     yLeftTop, 2 ) ) <= Math.pow( Utils.BALL_RADIUS, 2 )
190     || ( Math.pow( ballX - xRightBottom, 2) +
191     Math.pow( ballY - yRightBottom, 2 ) ) <=
192     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;
           else return CollisionType.SIDES;
166     }*/
167
168     //On se trouve dans le coin de droite en bas.
169     if( ballX - Utils.BALL_RADIUS <= xRightBottom && ballX -
170       Utils.BALL_RADIUS >= xLeftTop
171       && ballY - Utils.BALL_RADIUS <= yRightBottom
172       && ballY - Utils.BALL_RADIUS >= yLeftTop
173       )
174       return verifWhichSide( ballX, ballY, xRightBottom,
175         yRightBottom ) ? CollisionType.UPANDDOWN :
176         CollisionType.SIDES;
177     //On se trouve dans le coin de droite en haut.
178     else if( ballX - Utils.BALL_RADIUS <= xRightBottom && ballX
179       - Utils.BALL_RADIUS >= xLeftTop
180       && ballY + Utils.BALL_RADIUS <=
181         yRightBottom && ballY + Utils.
182         BALL_RADIUS >= yLeftTop )
183       return verifWhichSide( ballX, ballY, xRightBottom,
184         yLeftTop ) ? CollisionType.UPANDDOWN :
185         CollisionType.SIDES;
186     //On se trouve dans le coin de gauche en haut.
187     else if ( ballX + Utils.BALL_RADIUS <= xRightBottom && ballX
188       + Utils.BALL_RADIUS >= xLeftTop
189       && ballY + Utils.BALL_RADIUS <=
190         yRightBottom && ballY + Utils.
191         BALL_RADIUS >= yLeftTop )
192       return verifWhichSide( ballX, ballY, xLeftTop,
193         yLeftTop ) ? CollisionType.UPANDDOWN :
194         CollisionType.SIDES;
195     //On se trouve dans le coin de gauche en bas.
196     else if ( ballX + Utils.BALL_RADIUS <= xRightBottom && ballX
197       + Utils.BALL_RADIUS >= xLeftTop
198       && ballY - Utils.BALL_RADIUS <=
199         yRightBottom && ballY - Utils.
200         BALL_RADIUS >= yLeftTop )
201       return verifWhichSide( ballX, ballY, xLeftTop,
202         yRightBottom ) ? CollisionType.UPANDDOWN :
203         CollisionType.SIDES;
204     return null;
205 }
206
207 public boolean verifWhichSide( float xb, float yb, float xc, float
208   yc )
209 {
210     return ( Math.abs(( xb - xc ) / ( yb - yc )) <= 1 );
211 }
212
213 public void setBallPosition(float ballX, float ballY, float
214   ballSpeedX, float ballSpeedY) {
215     this.ballX = ballX;
216     this.ballY = ballY;
217     this.ballSpeedX = ballSpeedX;
218     this.ballSpeedY = ballSpeedY;
219 }
220 }

```

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.
25                     getInputStream()));
26                 out = new PrintWriter(socket.getOutputStream());
27                 Thread allocator = new Thread(new Allocator(in, out)
28                     );
29                 allocator.start();
30             }
31         } catch (IOException e) {
32             System.err.println("Erreur serveur à la réception !");
33         }
34     }
35
36     public static void main(String[] args){
37         int port = 2010;
38         if( args.length >= 1 ){
39             if( args[0].equals("-port") && args[1].length() > 0
40             ){
41                 port = Integer.valueOf(args[1]);
42             }
43         }
44         try{
45             System.out.println("Lancement du serveur en cours...
46                 ");
47             ServerBabyfoot s = new ServerBabyfoot(new
48                 ServerSocket(port));
49             System.out.println("Serveur prêt !");
50             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(Utills.
69                         SEPARATOR);
70                     String typeRequete = datas[0];
71                     if( typeRequete.equals("player") ){
72                         ServerBabyfoot.tplayer.
73                             setQuery(m);
74                         ServerBabyfoot.tplayer.
75                             handle(in, out);
76                     }else if( typeRequete.equals("match"
77                         ) ){
78                         ServerBabyfoot.tmatch.
79                             setQuery(m);
80                         ServerBabyfoot.tmatch.handle
81                             (in, out);
82                     }else if( typeRequete.equals("
83                         servers") ){
84                         ServerBabyfoot.tchat.
85                             setQuery(m);
86                         ServerBabyfoot.tchat.handle(
87                             in, out);
88                     }else if( typeRequete.equals("chat")
89                         ){
90                         ServerBabyfoot.tchat.
91                             setQuery(m);
92                         ServerBabyfoot.tchat.handle(
93                             in, out);
94                     }
95                 }else{
96                     break;
97                 }
98             }
99         } catch (IOException e) {
100             e.printStackTrace();
101         }
102     }
103 }

```