

Pong
Evolué

BEDIN Dylan
EPIARD Antoine
SCHIAPPARELLI Ugo

Enseignant de TD : MAZOIT Frédéric
Enseignant de cours : THIBAUT Samuel

Année universitaire 2015-2016

Table des matières

1	Introduction	2
2	Présentation des fonctionnalités	3
2.1	Multijoueur	3
2.1.1	Le réseau	4
2.2	Solo	5
2.3	Jeux de tests	6
3	Valorisation du travail réalisé	7
3.1	Prise en main sur une seule machine	7
3.2	Passage en réseau	8
3.3	Extension	9
3.3.1	Eviter la triche	9
3.3.2	Les balles spéciales et bonus	9
3.3.3	Menu	10
4	Diagramme des classes	11
5	Conclusion	12
6	Bibliographie	13
7	Annexes et code du projet	14
7.1	Main	15
7.2	MyThread	18
7.3	Ball	19
7.4	BallBonus	21
7.5	BallBonusLarger	23
7.6	BallBonusAccel	24
7.7	BallBonusMini	25
7.8	BallBonusSlow	26
7.9	BoxGameOver	27
7.10	Menu	28
7.11	Net	32
7.12	Pong	33

7.13	PongItem	44
7.14	Protected	45
7.15	Racket	46
7.16	Reseau	48
7.17	Tests	52
7.18	Window	56

Chapitre 1

Introduction

Ce rapport a pour ambition de présenter notre projet effectué dans le cadre de l'UE Réseau et Projet de Programmation 2. Il s'agit d'un Pong, l'un des tout premiers jeux vidéos, développé ici en Java et offrant la possibilité de jouer seul face à une IA ou en réseau, face à un autre joueur. Le Pong est une simulation de tennis de table. Ainsi, une balle rebondit entre les raquettes de 2 différents joueurs, situées aux extrémités gauche et droite de l'écran. Ceux-ci se renvoient ainsi mutuellement la balle jusqu'à ce que l'un d'eux marque un point en la faisant passer derrière la raquette de l'adversaire. La personne ayant marqué un point voit ainsi son score augmenter de 1. Les raquettes sont contrôlées à l'aide des touches directionnelles « haut » et « bas » et bougent de façon verticale sur l'écran. Nous allons ainsi par la suite présenter les différentes fonctionnalités de notre Pong, celles prévues originellement, la mise en place d'un pong multijoueur en réseau, mais aussi de nouvelles, développées par envies personnelles. Les fonctions et différents algorithmes composant ce projet seront ensuite explicités. Enfin, un diagramme des classes et une conclusion viendront résumer notre projet et permettront d'effectuer un bilan, traitant les limites et autres fonctionnalités supplémentaires qu'il serait envisageable d'implémenter.

Chapitre 2

Présentation des fonctionnalités

Lorsque l'on lance notre Pong, un menu s'affiche. Celui-ci présente 2 modes que l'on explicitera par la suite, le mode « Solo » et le mode « Multijoueur ». Le premier se présentant comme une extension au but du projet initial et possédant des règles spécifiques, nous allons d'abord nous concentrer sur le mode multijoueur.





2.1 Multijoueur

Le mode multijoueur est celui qui exploitera le réseau. Les 2 joueurs doivent lancer le Pong et l'un d'entre eux doit sélectionner « Serveur » dans le menu déroulant, après avoir choisi le mode multijoueur et la valeur du score à atteindre pour gagner la partie. La partie “nom du serveur” ne le concerne pas. Ainsi, son application se met en attente d'un client, soit le deuxième joueur. Ce dernier doit sélectionner « Client » et doit indiquer le nom de la machine sur laquelle il souhaite se connecter, soit son IP. La partie “score max” ne le concerne pas. Le jeu se lance alors en même temps pour les 2 joueurs.

Par défaut, le joueur de gauche est celui qui a sélectionné le serveur, et le client est le joueur de droite. Une balle apparaît au centre du jeu et se dirige arbitrairement sur lui. Par la suite, après chaque point marqué, le jeu se stoppe 1 seconde afin de marquer une pause entre chaque point et d'actualiser clairement le score, puis une nouvelle balle apparaît dans la direction du joueur ayant encaissé un point.

Cependant, le jeu ne s'arrête pas là. Un ensemble de « balles bonus » a été mis en place. Celles-ci ont des couleurs et une forme différentes, les distinguant de la balle de jeu originale. Elles apparaissent selon un rythme régulier, selon le score. En effet, le score des 2 joueurs est régulièrement additionné, et tous les x points (x correspondant à la valeur indiquée dans la colonne « Rythme » ci-dessous), la balle bonus correspondante apparaît au centre de l'écran et se dirige vers la

dernière personne ayant marqué un point s'il s'agit d'un bonus, son adversaire s'il s'agit d'un malus. Leurs effets respectifs disparaissent après 2 tours.

Image	Effet	Rythme
	Agrandit la raquette	5 pts
	Réduit la raquette	7 pts
	Accélère la raquette	11 pts
	Ralentit la raquette	14 pts

2.1.1 Le réseau

Le premier joueur est serveur lorsque l'on lance le jeu, cependant la communication se fait dans les 2 sens entre les joueurs, serveur et client. Tout d'abord, lorsque le premier joueur lance le jeu, il envoie la valeur de son score max au possible client qui se connecterait. Lorsque celui-ci arrive, il reçoit le score max et renvoie une chaîne de caractères servant d'accusé de réception (noté "ack" dans la suite du rapport). Ensuite, à chaque instant, chacun des joueurs envoie la position courante de sa raquette à l'adversaire qui peut ainsi positionner la raquette de son adversaire de son point de vue, afin d'être sûr que les 2 applications soient synchronisées. La balle est également synchronisée, cependant, pour pouvoir régulièrement vérifier si les valeurs envoyées sont les bonnes, les 2 joueurs gèrent la balle à tour de rôle. Lorsque la balle se situe sur la première moitié gauche de l'écran, le joueur 1 doit calculer la position et la vitesse de la balle et les envoyer au joueur 2 qui peut ainsi récupérer les valeurs et déplacer la balle sur son application à la position correspondante, ainsi que lui attribuer la bonne vitesse. Si la balle est sur la moitié droite de l'écran, c'est ainsi le joueur 2 qui gère la balle et le joueur 1 qui reçoit les informations, de façon similaire. Les raquettes et la balle ont également un système d'ack qui garantit l'envoi des données.

Les données sont envoyées selon un format particulier et suivent le protocole suivant. Une fonction "send" envoie les données, l'autre joueur les reçoit dans une fonction "read" et renvoie un ack qui sera réceptionné dans une fonction "ack". Si la fonction "ack" retourne "true", l'envoi et la réception se sont effectués correctement, sinon, on renvoie les données. Chacun des trois objets à synchroniser (les raquettes, la balle et le score max) le sont à l'aide des fonctions send/read différentes qui seront explicitées plus tard. La fonction "ack" quant à elle est générique et s'adapte à tous les objets. Voici le format des données qui sont

envoyées :

- *Coordonnées de la racket* : "x_racket"double ou "y_racket"double
- *Position de la balle* : "x_ballpos"double ou "y_ballpos"double
- *Vitesse de la balle* : "x_ballspeed"double ou "y_ballspeed"double
- *Score max* : "score"int

Ainsi, le nom de l'objet envoyé est un String, et est suivi de la donnée elle-même. La fonction read appelée correspondante effectue d'abord la fonction "readObject" afin de récupérer le String et vérifie si la valeur récupérée correspond à celle attendue. Si tel est le cas, la fonction "read" spécifique au type d'objet suivant est appelée et la variable à synchroniser est modifiée. Par la suite, la fonction read envoie le message servant d'ack qui est, selon le type de donnée envoyé, l'un des suivants :

- *Coordonnées de la racket* : "x_racket_recu" ou "y_racket_recu"
- *Ball position* : "x_ballpos_recu" ou "y_ballpos_recu"
- *Ball speed* : "x_ballspeed_recu" ou "y_ballspeed_recu"
- *Score max* : "score_recu"

L'envoi de chacune des données est contenue dans une boucle "do while" de la sorte :

```
do{
    r.send(sock, ballData[0], "x_ballpos");
}while(!r.ack(sock, "x_ballpos_recu"));
do{
    r.send(sock, ballData[0], "y_ballpos");
}while(!r.ack(sock, "y_ballpos_recu"));
do{
    r.send(sock, ballData[1], "x_ballspeed");
}while(!r.ack(sock, "x_ballspeed_recu"));
do{
    r.send(sock, ballData[1], "y_ballspeed");
}while(!r.ack(sock, "y_ballspeed_recu"));
```

Ainsi, l'envoi s'est effectué correctement si la chaîne de caractère souhaitée en ack, celle passée en paramètre, a été envoyée, la fonction "ack" retournant donc "true". Sinon, on renvoie les données.

2.2 Solo

Nous avons jugé intéressant de ne pas seulement faire un mode multijoueur mais également d'intégrer un mode solo, qui donc ne fait forcément pas appel au réseau, auquel nous avons intégré des règles spécifiques. Pour le lancer, il suffit de sélectionner "Solo" dans "Mode de jeu" et de lancer la partie à l'aide de "Play". Les options suivantes ne servent qu'en mode "Multijoueur". Le joueur se retrouve face à une raquette, contrôlée par l'IA, imbattable. Il est en effet impossible pour lui de marquer un point comme dans le mode multijoueur, car la raquette en face calcule la coordonnée y de la balle et se positionne selon

elle, lui permettant ainsi de la réceptionner en permanence. Le score du joueur est ainsi calculé par le nombre de fois où il arrive à renvoyer la balle. Son score est augmenté de 1 à chaque rebond et repart à 0 lorsque celui-ci se prend un point. Ce mode solo peut ainsi être vu comme un mode entraînement.

2.3 Jeux de tests

Nos tests sont situés dans la classe `Tests.java` et on peut les effectuer en lançant le programme avec pour argument la chaîne de caractères `"test"`. Ici, le menu ne se lancera pas mais les tests seront fait automatiquement et l'utilisateur verra s'afficher sur la sortie standard le message suivant : `"Les tests ont été passés avec succès"`, si les tests se passent sans problèmes. Nous avons décidé de créer des tests sur des fonctions jugées fondamentales et complexes, afin d'être sûr que celles-ci agissent de la façon attendue, et ce quels que soient les cas. Nous avons ainsi choisi les fonctions `"collision"` et `"rebound"`, situées toutes deux dans la classe `"Pong.java"`.

La méthode est similaire pour les 2. Nous lançons un parcours composé de 4 boucles `"for"`, la première sélectionnant la première puis la deuxième raquette, la deuxième modifiant la trajectoire `y` de la raquette courante et les 2 suivantes bougeant la balle dans l'espace en `x` et `y`. Ainsi, nous testons toutes les positions possibles de la balle en fonction de toutes les positions possibles de chaque raquette, indépendamment l'une de l'autre. Dans chaque cas, nous vérifions si la fonction à tester retourne bien les paramètres attendus. Nous comparons ainsi les résultats dits `"expérimentaux"`, correspondant aux valeurs réelles, celles obtenues de par les objets `"Pong"` et `"Ball"` créés, avec les résultats attendus, c'est-à-dire les valeurs souhaitées dans les fonctions tests.

Chapitre 3

Valorisation du travail réalisé

Ici, nous allons mettre en valeur notre code en explicitant les différents algorithmes utilisés et notre réflexion ayant mené à ce projet final. Pour cela, nous allons notamment voir l'exemple d'un bug que l'on a eu, la façon dont on l'a résolu, ainsi que les optimisations ayant permis d'éviter au maximum les redondances dans le code.

3.1 Prise en main sur une seule machine

La première étape du projet a consisté à créer un pong basique sans le réseau. Nous ne sommes pas parti de rien, le sujet étant donné avec du code. Nous avons pu apprendre à nous servir de swing, l'API de java pour faire une interface, et commencer la première partie qui a consisté à créer un pong multijoueur, jouable sur une seule machine. Notre première décision fut de changer l'aspect graphique du jeu, se rapprochant plus du pong originel.

Nous avons repris l'implémentation donnée dans le code de base et le sujet, une classe pong qui gère tout le jeu, une classe Window qui affiche le rendu et par laquelle l'appel au réseau, contenu dans une classe à part, se fera plus tard. Une fonction Main lance le jeu et décide ce que feront le serveur et le client. Enfin, il y a deux classes, Racket et Ball, héritant toutes deux de PongItem, interface contenant 3 paramètres communs aux 2 objets, "width", "height" et "position". Le premier problème fut de gérer la collision et le rebond de la balle sur les raquettes. Pour cela, nous avons utilisé un algorithme prenant le problème de revers, c'est-à-dire que nous avons cherché à savoir si la balle n'était pas sur la raquette. En pensant différemment, nous avons donc pu contourner le problème. Le second, le problème lié au rebond, était dû à la différenciation de l'axe x ou l'axe y sur lesquels la balle devait rebondir. Avec notre nouveau mode de pensée utilisé et appliqué sur le rebond, nous avons pu résoudre ce problème aussi. Plus tard, dans le projet, nous avons de nouveau modifié le rebond pour avoir quelque chose de plus complexe, nous avons coupé la raquette en cinq et, selon la partie sur laquelle la balle rebondit, l'orientation de ce dernier et la vitesse

sera différente.

3.2 Passage en réseau

Une fois le mode de jeu multijoueur mis en place de façon locale, il aura fallu mettre en place le réseau. Pour cela, comme expliqué précédemment, une classe Réseau a été créée, contenant 2 fonctions, send et read. L’envoi et la réception de données et le système d’ack ayant déjà été détaillé, nous allons ici nous concentrer sur le mode d’envoi de données choisi, la raison de la multiplicité des fonctions read et send ainsi que celle de l’unicité de la fonction ack.

Après la rédaction de plusieurs fonctions send et read, afin d’envoyer et recevoir les données de la racket, de la balle et du score max à établir, nous avons réussi à factoriser le tout en obtenant 4 fonctions. Ainsi, les fonctions “read” et “send” sont là pour envoyer des coordonnées. Il peut s’agir des coordonnées de position de la racket, de position de la balle ou de vitesse de celle-ci. Voici les prototypes de ces 2 fonctions :

- *public void send(Socket sock, Point point, String coor)*
- *public void read(Socket sock)*

Les 2 fonctions prennent ainsi nécessairement la socket pour l’envoi/réception des données. La fonction “send”, quant à elle, prend la variable “point” ainsi que le String “coor” déterminant l’origine du point ainsi que la coordonnée de celui-ci à envoyer.

Les 2 autres fonctions send/read sont les suivantes :

- *public void sendMaxScore(Socket sock, int scoreMax)*
- *public void readMaxScore(Socket sock)*

Celles-ci sont appelées une seule fois au début de la partie, contrairement aux 2 précédentes, mais fonctionnent sur le même principe, seul le type de donnée diffère. Ici, il s’agit d’un int. Une fonction readMaxScore spécifique a donc été créée afin de réceptionner la valeur correctement.

La fonction ack, intégrée dans un do while, comme expliqué précédemment, est générique et s’applique à absolument tout type de donnée, à condition qu’il respecte ces différentes conditions. Tout d’abord, lors du read, la dernière étape doit être l’envoi d’une chaîne de caractère unique servant à identifier le type de donnée qui a été réceptionné. Ensuite, cette chaîne de caractère doit être passée en paramètre à la fonction “ack” qui la comparera avec celle reçue et renverra “True” si se sont les mêmes, “False” sinon, entraînant l’envoi de l’information à nouveau. Voici le prototype de cette fonction :

- *public boolean ack(Socket sock, String message)*

Cette fonction récupère ainsi les informations via la socket et compare la chaîne de caractères obtenue avec la chaîne “message” passée en paramètre.

3.3 Extension

3.3.1 Eviter la triche

Pour éviter la triche, nous avons repris l'algorithme et les explications données dans le sujet. On calcule ainsi la position attendue par la balle à l'aide de sa vitesse, puis on la compare avec celle reçue. Si ce sont les mêmes, il n'y a pas de triche. On ajoute l'appel à cet algorithme après les boucles traitant la réception des données.

Plus tard, nous avons ajouté des balles bonus qui augmentent ou diminuent la vitesse. Il fallait un signal pour avertir que les raquettes étaient affectées par les bonus. Grâce à notre implémentation, nous avons un paramètre de type `int` dans la structure `racket` indiquant si une raquette est touchée par un bonus. Il suffit donc de vérifier si cet `int` était à la bonne valeur pour les deux raquettes.

3.3.2 Les balles spéciales et bonus

La seconde extension que nous avons ajoutée fut les bonus. Nous avons commencé par un bonus agrandissant la raquette courante du joueur, en prenant en compte le fait qu'il faut gérer une collision différente. Il faut donc se servir de notre fonction implémentée pour les collisions avec en argument `ball` et `racket`. Les bonus étant tous des types de `Ball` particuliers, ils héritent tous de cette classe.

Ne voulant pas surcharger le réseau de données, nous avons décidé que chaque jeu gèrera lui-même tous les bonus et qu'ils apparaîtraient donc au bout d'un certain nombre de points marqués par les deux joueurs. Ainsi, on garantit la synchronisation, le bonus arrivant sur la raquette ayant marqué le dernier point. Nous avons créé plusieurs données membres pour une classe nommée `BallBonusLarger`. Tout d'abord, un booléen qui nous permet de savoir si le bonus a été créé, qui sert afin de ne pas écrire de données sur des bonus non instanciés, grâce à un `if`. Puis, un tableau de 2 `int`, qui correspond au score courant de la partie lors de la création du bonus. Cette donnée nous sert à ne pas créer de nouveaux bonus lorsqu'un est déjà instancié et que le score contenu dans le tableau correspond au score où celui-ci a été créé. Cela évite que les bonus apparaissent sans cesse tant que le score n'est pas modifié. Enfin, un entier qui définit la position Y du bonus, qui correspondra à la hauteur de la dernière balle perdue.

Après cela, nous avons créé de nouveaux bonus. L'un d'entre eux réduit la raquette du joueur qui l'obtient, tandis que les autres augmentent sa vitesse ou au contraire, la réduisent. Pour ce faire, nous avons créé une classe, située au-dessus dans la hiérarchie, nommée `BallBonus`, héritant de `Ball`. Ainsi, toutes les classes des différents bonus héritent de `BallBonus`. Puis, nous avons mis dans cette classe ce que nous avons précédemment fait dans `BallBonusLarger` avec un constructeur permettant de choisir l'image du bonus. Enfin, nous avons créé des fonctions utilisées lors de la création et l'affichage, utilisant des données membres pouvant être changées par le constructeur, nous permettant de gérer différents paramètres aisément, comme le nombre de points à marquer avant

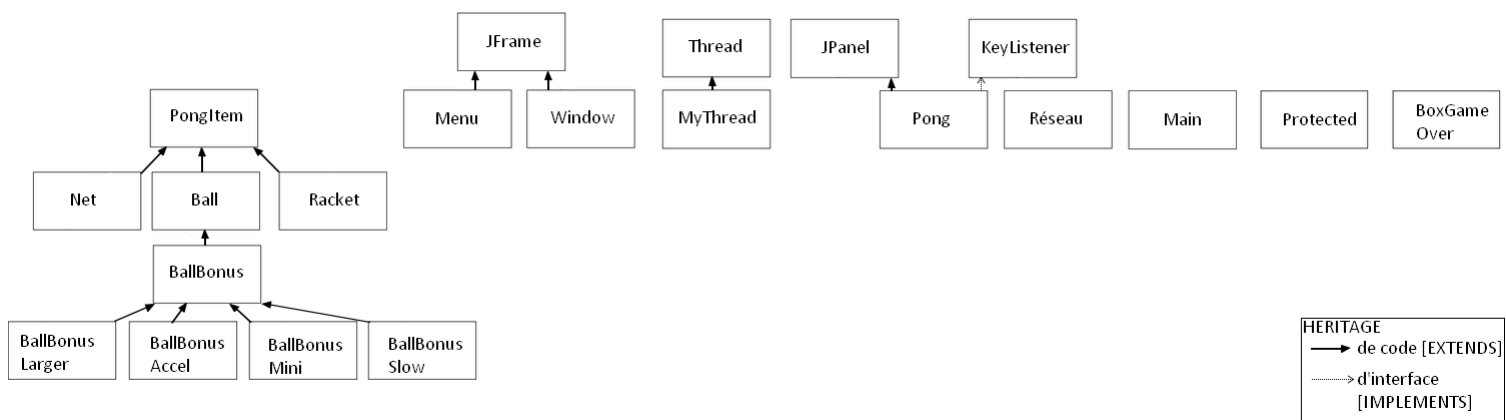
que le bonus apparaisse, l'effet que le bonus aura sur la raquette et le sens dans lequel il ira. Il se déplacera ainsi sur celui qui a marqué le dernier point s'il s'agit d'un bonus considéré comme positif, ou l'adversaire s'il s'agit d'un malus.

3.3.3 Menu

En dernière partie, nous avons décidé d'ajouter un menu qui nous faciliterait, autant à nous, programmeurs, qu'à l'utilisateur, le lancement d'une partie de pong. Après des recherches pour comprendre la façon dont swing gère les menus, nous avons donc créé une classe Menu. Pour intégrer les données postées par l'utilisateur dans le menu au reste du code, nous avons créé des données membres, dans la classe Main, accessibles par toutes les classes permettant de gérer les différentes options du menu. Nous avons aussi créé un thread, que nous avons utilisé dans le Main, pour attendre que l'utilisateur lance le jeu avant de commencer à lancer le réseau, puisqu'il nous fallait les réponses de celui-ci. Une des options du menu en début de partie est le score max que doit atteindre un joueur pour gagner une partie, et donc il fallait aussi gérer les joueurs qui ne mettraient pas la même valeur, alors qu'ils jouent ensemble. Pour contourner ce problème, nous avons décidé que ce serait le serveur qui choisirait le score max et qui l'enverrait en début de partie au client. Enfin, nous avons créé une boîte de dialogue pour la fin d'une partie, lancée lorsqu'un joueur atteint le score maximum et proposant deux options : une permettant de rejouer, l'autre de fermer la fenêtre du jeu. La seule chose à réinitialiser pour une nouvelle partie fût le score, et pour la fermeture de la fenêtre, on envoie un entier qui nous permet de sortir de la boucle while de Window, là où toute la partie affichage se fait.

Chapitre 4

Diagramme des classes



Chapitre 5

Conclusion

Le bilan de ce projet est globalement positif. Nous pensons avoir réussi, de manière efficace, à mettre en place un pong en réseau, le tout en lui ajoutant de nombreux éléments facultatifs, tels que la mise en place du menu, le mode solo ou les balles bonus.

Cependant, lors de la création de celui-ci, nous avons eu de nombreuses idées d'implémentations supplémentaires qui n'ont pas été ajoutées, faute de temps et par volonté de se concentrer sur des aspects plus fondamentaux. Parmi ces fonctionnalités, il y a la possibilité de jouer à quatre joueurs simultanément. Il aurait donc fallu modifier notre réseau pour que toutes les machines se connectent entre elles et soient à la fois serveur et client. Dès lors qu'une machine se connecterait à une autre, celle-ci lui enverrait les noms des autres machines auxquelles il faudrait se connecter. On aurait donc pu gérer la balle non plus sur deux machines, mais sur quatre,

et il aurait fallu modifier le code du jeu en rajoutant deux raquettes, en haut et en bas, avec un score différent pour chacun. Nous aurions aussi aimé pouvoir griser les menus lorsque ceux-ci sont inutiles, par exemple, lorsque l'on est serveur, l'option demandant l'adresse ip est inutile, nous aurions donc voulu que le menu soit dynamique et change dès que l'on choisit une option. Cependant, nous avons évidemment adapté l'implémentation, de façon à ce que, si un joueur étant serveur tape une adresse IP ou que le client souhaite déterminer le score à atteindre lui-même, ces valeurs ne soient pas prises en compte. Enfin, d'autres idées de bonus nous ont plu et seront probablement implémentées à l'issue de ce projet, comme le fait de jouer avec plusieurs balles, obtenir une raquette aimantée bloquant la balle quelques secondes, un malus inversant les touches, un bonus offrant une possibilité de mouvement selon l'axe x temporairement etc..

Enfin, nous ne garantissons pas aux joueurs une protection face à tout type de triche, et ce malgré notre algorithme, même s'il paraît bien sûr impossible de faire cela. Nous pensons malgré tout que celui-ci peut-être améliorable, bien que nous considérons néanmoins qu'il offre au joueur une garantie correcte au vu de l'ampleur du projet.

Chapitre 6

Bibliographie

Voici les différents supports nous ayant aidé pendant le développement du projet. Il s'agit de sites web, ainsi, ceux-ci peuvent avoir été modifiés depuis leur consultation, pendant la durée du projet.

Cours par M. Cyrille HERBY pour l'utilisation de swing : <https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-menus-et-boites-de-dialogue>

Documentation Java Oracle pour diverses informations sur les fonctions (notamment pour la classe réseau) : <https://docs.oracle.com/javase/7/docs/api/>

Support de cours de M. Samuel THIBAULT : <http://dept-info.labri.fr/thibault/Reseau/support.pdf>

Chapitre 7

Annexes et code du projet

Cette annexe aura pour objectif de présenter l'ensemble des documents liés au projet, à commencer par le sujet.

7.1 Main

```
package pong;

import java.net.*;
import java.awt.Point;
import java.io.*;

import pong.gui.Ball;
import pong.gui.Menu;
import pong.gui.Window;
import pong.gui.Pong;
import pong.gui.Tests;

/**
 * Starting point of the Pong application
 */

public class Main {

    public static Point ballPosition, ballSpeed;
    public static Point currentRacket, otherRacket;
    private static int indice = 0, indiceSC = -1, nbPlayer, scoreMax = 15;
    private static String serverName;

    public static void main(String[] args) throws IOException {
        //Test
        if(args.length == 1){
            if(args[0].equals("test")){
                Tests test = new Tests();
                //-1 = erreur
                if(test.TestsCollision() == -1)
                    return;
                if(test.TestsRebound() == -1)
                    return;

                System.out.println("Les tests ont ete passes avec succes");
                return;
            }
        }
        else{
            Menu menu = new Menu();
            MyThread thread = new MyThread();
            thread.start();
            while(thread.isAlive()){

                //Parametres de base pour la balle
                ballPosition = new Point(500, 300);
                ballSpeed = new Point(Ball.BALL_SPEED, Ball.BALL_SPEED);
            }
        }
    }
}
```

```

//Mode multijoueur
if (getNbPlayer() == 2){
    //Serveur
    if (getIndiceSC() == 0) {
        ServerSocket secoute = new ServerSocket(7777);
        currentRacket = new Point(50, 0);
        otherRacket = new Point(930, 0);
        Socket client = secoute.accept();
        client.setTcpNoDelay(true);
        Pong pong = new Pong(1, 2);
        Window window = new Window(pong);
        window.displayOnscreen(client);
        secoute.close();
    }
    //Client
    else if (getIndiceSC() == 1){
        currentRacket = new Point(930, 0);
        otherRacket = new Point(50, 0);
        Socket s = new Socket(getServerName(), 7777);
        s.setTcpNoDelay(true);
        if (s.isConnected()) {
            Pong pong = new Pong(2, 1);
            Window window = new Window(pong);
            window.displayOnscreen(s);
        }
        s.close();
    }
}

//Mode solo
else if (getNbPlayer() == 1){
    Pong pong = new Pong(1, 0);
    Window window = new Window(pong);
    window.displayOnscreen();
}
}

//Getters/Setters
public static int getIndice() {
    return indice;
}

public static void setIndice(int indice) {
    Main.indice = indice;
}

public static int getIndiceSC() {
    return indiceSC;
}

```

```
public static void setIndiceSC(int indiceSC) {
    Main.indiceSC = indiceSC;
}

public static int getNbPlayer() {
    return nbPlayer;
}

public static void setNbPlayer(int nbPlayer) {
    Main.nbPlayer = nbPlayer;
}

public static String getServerName() {
    return serverName;
}

public static void setServerName(String serverName) {
    Main.serverName = serverName;
}

public static int getScoreMax() {
    return scoreMax;
}

public static void setScoreMax(int scoreMax) {
    Main.scoreMax = scoreMax;
}
}
```

7.2 MyThread

```
package pong;

public class MyThread extends Thread{

    public void run(){
        while(Main.getIndice() == 0){//Tant que la variable indice dans
            Main vaut 0, le thread existe (celui -ci est modifie dans
            menu.java)
            try {
                Thread.sleep(500);
            }
            catch (InterruptedException ex) {}
        }
    }
}
```

7.3 Ball

```
package pong.gui;

import java.awt.Image;
import java.awt.Point;
import java.awt.Toolkit;
import javax.swing.ImageIcon;

public class Ball extends PongItem{

    /**
     * Speed of ball (in pixels per second)
     */
    public static final int BALL_SPEED = 3;
    protected Image image;
    protected Point speed = new Point(BALL_SPEED, BALL_SPEED);
    protected int lastPositionY = 0;
    private int lastTouch = 1; //Derniere raquette ayant touche la balle

    public Ball NextPosition(){
        return new Ball(this.position.x+this.speed.x,
            this.position.y+this.speed.y);
    }

    public int getLastPositionY(){
        return this.lastPositionY;
    }

    public void setLastPositionY(int p){
        this.lastPositionY = p;
    }

    //Trois constructeurs differents pour 3 parametres differents image
    et position (x, y)

    public Ball(){
        ImageIcon icon;
        this.image = Toolkit.getDefaultToolkit().createImage(
            ClassLoader.getResource("image/ball.png"));
        //Chargement d'image
        icon = new ImageIcon(image);
        this.width = icon.getIconWidth();
        this.height = icon.getIconHeight();
        this.position = new Point(500, 300);
    }
}
```

```

public Ball(int x, int y){
    ImageIcon icon;

    this.image = Toolkit.getDefaultToolkit().createImage(
        ClassLoader.getResource("image/ball.png"));
    icon = new ImageIcon(image);
    this.width = icon.getIconWidth();
    this.height = icon.getIconHeight();
    this.position = new Point(x, y);
}

public Ball(int x, int y, String im){
    ImageIcon icon = null;
    this.image = Toolkit.getDefaultToolkit().createImage(
        ClassLoader.getResource(im));
    icon = new ImageIcon(this.image);
    this.width = icon.getIconWidth();
    this.height = icon.getIconHeight();
    this.position = new Point(x, y);
}

public int getLastTouch(){
    return this.lastTouch;
}

public void setLastTouch(int player){
    this.lastTouch = player;
}
}

```

7.4 BallBonus

```
package pong.gui;

public class BallBonus extends Ball{

    protected int bonus_speed = -5;
    private boolean instance = false;
    private int[] scoreHist; //Variable indiquant le score des joueurs au
        moment ou une BallBonus est cree
    private int positionY = 0;
    private int frequence;
    private boolean malus;
    private int numBonus;

    public BallBonus(String im, int frequence, boolean malus, int
        numBonus){
        super(500, 20, im);
        scoreHist = new int[2];
        scoreHist[0] = 0;
        scoreHist[1] = 0;
        this.frequence = frequence;
        this.malus = malus;
        this.numBonus = numBonus;
    }

    public int getSpeed(){
        return this.bonus_speed;
    }

    public boolean getInstance(){
        return this.instance;
    }

    public void setInstance(boolean b){
        this.instance = b;
    }

    public int getBonusSpeed(){
        return this.bonus_speed;
    }

    public void setBonusSpeedLeft(){
        if (!malus)
            this.bonus_speed = -5;
        else
            this.bonus_speed = 5;
    }
}
```

```

public void setBonusSpeedRight(){
    if(!malus)
        this.bonus_speed = 5;
    else
        this.bonus_speed = -5;
}

public int getPositionY(){
    return this.positionY;
}

public boolean isHist(int x, int y){
    return this.scoreHist[0] == x && this.scoreHist[1] == y;
}

public void setHist(int p1, int p2){
    this.scoreHist[0] = p1;
    this.scoreHist[1] = p2;
}

public void deformationRacket(Racket rack){ //indique ce qu'il faut
    faire a la raquette lorsqu'elle entre en collision avec le bonus
    if(this.numBonus == 0)//BallBonusLarger
        rack.setImage("./image/racketLarger.png");
    if(this.numBonus == 1)//BallBonusMini
        rack.setImage("./image/racketMini.png");
    if(this.numBonus == 2){//BallBonusAccel
        if(rack.position.x == 50)//On differencie raquette droite et
            gauche
            Pong.RACKET_SPEED_JG = Pong.RACKET_SPEED_JD * 2;
        else
            Pong.RACKET_SPEED_JD = Pong.RACKET_SPEED_JD * 2;
    }
    if(this.numBonus == 3){//BallBonusSlow
        if(rack.position.x == 50)//On differencie raquette droite et
            gauche
            Pong.RACKET_SPEED_JG = Pong.RACKET_SPEED_JG / 4;
        else
            Pong.RACKET_SPEED_JD = Pong.RACKET_SPEED_JG / 4;
    }
}

public boolean apparition(int score1, int score2){ //indique quand
    apparait la balle bonus
    return (score1 + score2) % this.frequence == 0;
}
}

```

7.5 BallBonusLarger

```
package pong.gui;

public class BallBonusLarger extends BallBonus{

    public BallBonusLarger(){
        super("image/ball_larger.png", 5, false, 0); //Image du bonus,
            frequence, malus, numBonus
    }
}
```

7.6 BallBonusAccel

```
package pong.gui;

public class BallBonusAccel extends BallBonus{

    public BallBonusAccel(){
        super("image/ball_accel.png", 11, false, 2); //Image du bonus,
            frequence, malus, numBonus
    }

}
```

7.7 BallBonusMini

```
package pong.gui;

public class BallBonusMini extends BallBonus{

    public BallBonusMini(){
        super("image/ball_mini.png", 7, true, 1); //Image du bonus,
            frequence, malus, numBonus
    }

}
```

7.8 BallBonusSlow

```
package pong.gui;

public class BallBonusSlow extends BallBonus{

    public BallBonusSlow(){
        super("image/ball_slow.png", 14, true, 3); //Image du bonus,
            frequence, malus, numBonus
    }
}
```

7.9 BoxGameOver

```
package pong.gui;

import javax.swing.*.*;

public class BoxGameOver {

    public int BoxGameOver(Pong pong, Window window, int joueur){
        JOptionPane jop = new JOptionPane(); //Boite de dialogue
        String joueurW = Integer.toString(joueur); //Passage en string du
            joueur gagnant
        int option = jop.showConfirmDialog(null,"Joueur " + joueurW + " a
            gagne. Voulez-vous rejouer ?", "Jeu fini",
            JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
        if(option == JOptionPane.OK_OPTION){ //Si on clique sur OK
            int[] newTab = new int[2];
            newTab[0] = 0;
            newTab[1] = 0;
            pong.setScore(newTab); //reinitialisation du score
        }
        if(option == JOptionPane.NO_OPTION){ //Si on clique sur non
            window.dispose(); //On ferme la fenetre
            return 1; //On retourne un int servant de signal
        }
        return -1;
    }
}
```

7.10 Menu

```
package pong.gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;

import pong.Main;

public class Menu extends JFrame {

    private static final long serialVersionUID = 1L;
    private JLabel nomServLabel, nbPlayerLabel, ClSeLabel, endGameLabel;
    //Les differents menu
    private JComboBox nbPlayer, ClSe; //Type de menu
    private JTextField nomServ, endGame; //Type de menu

    public Menu(){
        this.setTitle("Pong");
        this.setSize(300, 350);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLocationRelativeTo(null);
        this.initComponent();
        this.setVisible(true);
    }

    private void initComponent(){

        //Solo ou multijoueur
        JPanel panNbPlayer = new JPanel();
```

```

panNbPlayer.setBackground(Color.white);
panNbPlayer.setPreferredSize(new Dimension(220, 60));
panNbPlayer.setBorder(BorderFactory.createTitledBorder("Mode de
    jeu"));
nbPlayer = new JComboBox();
nbPlayer.addItem("Solo");
nbPlayer.addItem("Multijoueur");
nbPlayerLabel = new JLabel("Mode : ");
panNbPlayer.add(nbPlayerLabel);
panNbPlayer.add(nbPlayer);

//Client ou serveur
JPanel panClSe = new JPanel();
panClSe.setBackground(Color.white);
panClSe.setPreferredSize(new Dimension(220, 60));
panClSe.setBorder(BorderFactory.createTitledBorder("Reseau"));
ClSe = new JComboBox();
ClSe.addItem("Serveur");
ClSe.addItem("Client");
ClSeLabel = new JLabel("Reseau : ");
panClSe.add(ClSeLabel);
panClSe.add(ClSe);

//Le nom du client
JPanel panNomServ = new JPanel();
panNomServ.setBackground(Color.white);
panNomServ.setPreferredSize(new Dimension(220, 60));
nomServ = new JTextField();
nomServ.setPreferredSize(new Dimension(100, 25));
panNomServ.setBorder(BorderFactory.createTitledBorder("Nom du
    serveur adverse"));
nomServLabel = new JLabel("Saisir un nom :");
panNomServ.add(nomServLabel);
panNomServ.add(nomServ);

//score pour finir la partie
JPanel panEndGame = new JPanel();
panEndGame.setBackground(Color.white);
panEndGame.setPreferredSize(new Dimension(220, 60));
endGame = new JTextField();
endGame.setPreferredSize(new Dimension(100, 25));
panEndGame.setBorder(BorderFactory.createTitledBorder("Score
    max"));
endGameLabel = new JLabel("Saisir un chiffre");
panEndGame.add(endGameLabel);
panEndGame.add(endGame);

JPanel content = new JPanel();
content.setBackground(Color.white);
content.add(panNbPlayer);

```

```

content.add(panClSe);
content.add(panNomServ);
content.add(panEndGame);

JPanel control = new JPanel();
JButton okBouton = new JButton("Play");

okBouton.addActionListener(new ActionListener(){ //Bouton
    public void actionPerformed(ActionEvent arg0) { //si il est
        appuye, on associe les valeurs aux variable dans main
        Main.setIndice(1);
        Main.setIndiceSC(getClSe());
        Main.setNbPlayer(getSoloMulti());
        if (Main.getIndiceSC() == -1){
            System.out.println("Probleme avec indice-Serveur-Client");
        }
        Main.setServerName((String)nomServ.getText());
        String scoreMaxString = (String)endGame.getText();
        //Serveur
        if(!scoreMaxString.equals("") &&
            Integer.parseInt(scoreMaxString) != 0){
            Main.setScoreMax(Integer.parseInt(scoreMaxString));
        }
        setVisible(false);
    }
});

JButton cancelBouton = new JButton("Annuler"); //Bouton annuler
cancelBouton.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent arg0) {
        setVisible(false);
    }
});

control.add(okBouton);
control.add(cancelBouton);

this.getContentPane().add(content, BorderLayout.CENTER);
this.getContentPane().add(control, BorderLayout.SOUTH);
}

//Change les string en int
private int getClSe(){
    String stringTest = (String)ClSe.getSelectedItemAt();
    if(stringTest.equals("Serveur")){
        return 0;
    }
    else if (stringTest.equals("Client")){

```



```
        return 1;
    }
    return -1;
}

private int getSoloMulti(){
    String test = (String)nbPlayer.getSelectedItem();
    if(test.equals("Solo")){
        return 1;
    }
    else if(test.equals("Multijoueur")){
        return 2;
    }
    return -1;
}
}
```

7.11 Net

```
package pong.gui;

import java.awt.Image;
import java.awt.Toolkit;

import javax.swing.ImageIcon;

public class Net extends PongItem{
    /*
     * Filet apparaissant au milieu de l'ecran
     */

    protected final Image image;

    public Net(){ //Image apparaissant au milieu de l'ecran
        ImageIcon icon;

        this.image = Toolkit.getDefaultToolkit().createImage(
            ClassLoader.getResource("image/net.png"));
        icon = new ImageIcon(image);
        this.width = icon.getIconWidth();
        this.height = icon.getIconHeight();
    }
}
```

7.12 Pong

```
package pong.gui;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Dimension;
import java.awt.Point;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.lang.Math;

import javax.swing.JPanel;

import pong.Main;

/**
 * An Pong is a Java graphical container that extends the JPanel class in
 * order to display graphical elements.
 */
public class Pong extends JPanel implements KeyListener {

    private static final long serialVersionUID = 1L;

    /**
     * Constant (c.f. final) common to all Pong instances (c.f. static)
     * defining the background color of the Pong
     */
    private static final Color backgroundColor = new Color(0);

    /**
     * Width of pong area
     */
    private static final int SIZE_PONG_X = 1000;

    /**
     * Height of pong area
     */
    private static final int SIZE_PONG_Y = 600;

    /**
     * Time step of the simulation (in ms)
     */
    public static final int timestep = 10;

    /**
     * Pixel data buffer for the Pong rendering
     */
    private Image buffer = null;
```

```

/**
 * Graphic component context derived from buffer Image
 */
private Graphics graphicContext = null;
/**
 * Speed of racket (in pixels per second)
 */
public static int RACKET_SPEED_JD = 15;//Vitesse de la raquette droite

public static int RACKET_SPEED_JG = 15;//Vitesse de la raquette gauche

private int nbPlayer = 2;

private Racket[] racket = new Racket[2];

private int posOtherRacket = 0;

private int nbBonus = 4;

private BallBonus[] tabBallBonus = new BallBonus[nbBonus];

private Ball ball;

private Net net;

/*
 * score[0] = score du joueur 1
 * score[1] = score du joueur 2
 */
private int[] score;

public Pong(int currentRacketIndice, int otherRacketIndice){
    this.getRacket()[0] = new Racket(currentRacketIndice);
    this.getRacket()[1] = new Racket(otherRacketIndice);
    this.setBall(new Ball());
    this.setPreferredSize(new Dimension(SIZE_PONG_X, SIZE_PONG_Y));
    this.addKeyListener(this);
    this.setScore(new int[2]);
    this.getScore()[0] = 0;
    this.getScore()[1] = 0;
    this.net = new Net();
    this.tabBallBonus[0] = new BallBonusLarger();
    this.tabBallBonus[1] = new BallBonusMini();
    this.tabBallBonus[2] = new BallBonusAccel();
    this.tabBallBonus[3] = new BallBonusSlow();
    if(otherRacketIndice == 0){
        this.nbPlayer = 1;
    }
}

```

```

public int getPlayerScore(int player){
    return this.getScore()[player - 1];
}

protected Point getPositionBall(){
    return this.getBall().position;
}

public Point getCurrentRacketPosition(){
    return this.getRacket()[0].position;
}

public Point getBallPosition(){
    return getBall().position;
}

public Point getSpeedPosition(){
    return getBall().speed;
}

public int getSpeedRacket(){
    return this.getRacket()[0].speed;
}

public int[] getScore() {
    return score;
}

public void setScore(int[] score) {
    this.score = score;
}

public boolean ballMyTurn(){
    //Si la raquette courrante est la raquette de gauche
    if(getRacket()[0].position.getX() == 50){
        return getBallPosition().getX() <= SIZE_PONG_X / 2;
    }
    else{
        return getBallPosition().getX() > SIZE_PONG_X / 2;
    }
}
/**
    * Proceeds to the movement of the ball and updates the screen
    */
public void animate() {
    /* Mise a jour de la position des raquettes */
    if(!collision(getBall().NextPosition(),
        getRacket()[0].NextPosition())){ //Notre raquette
        getRacket()[0].position.y += getRacket()[0].speed;
    }
}

```

```

    /* Limite de l'ecran pour la raquette*/
    if (getRacket()[0].position.y < 0)
        getRacket()[0].position.y = 0;
    if (getRacket()[0].position.y > SIZE_PONG_Y -
        getRacket()[0].height)
        getRacket()[0].position.y = SIZE_PONG_Y -
            getRacket()[0].height;
}

if(this.nbPlayer == 2){
    /*Mise a jour de la raquette adverse*/
    getRacket()[1].position.x = (int) Main.otherRacket.getX();
    getRacket()[1].position.y = (int) Main.otherRacket.getY();

    /* Mise a jour de la balle*/
    if(ballMyTurn()){ //Si c'est mon tour
        for(int i = 0; i < 2; i++){
            if(collision(getBall().NextPosition(), getRacket()[i])){
                //Et qu'il y a collision
                this.rebound(i); //on fait rebondir la balle
            }
        }
    }
    /*Si ce n'est pas notre tour*/
    else{
        getBall().position.x = (int) Main.ballPosition.getX();
        getBall().position.y = (int) Main.ballPosition.getY();
        getBall().speed.x = (int) Main.ballSpeed.getX();
        getBall().speed.y = (int) Main.ballSpeed.getY();
    }
}

if(this.nbPlayer == 1){//Mode solo
    /*Mise a jour de la position de la raquette ennemie*/
    getRacket()[1].position.y = getBall().NextPosition().position.y
        - this.posOtherRacket + 1;
    if(getRacket()[1].position.y < 0){
        getRacket()[1].position.y = 0;
    }
    if(getRacket()[1].position.y > 530){
        getRacket()[1].position.y = 530;
    }
    for(int i = 0; i < 2; i++){
        if(collision(getBall().NextPosition(), getRacket()[i])){
            this.rebound(i);
            if(i == 1){
                //Aleatoire de la position de la balle sur la raquette
                (raquette droite)
                this.posOtherRacket = getPosOtherRacket();
            }
        }
    }
}

```

```

        else{
            //Le score augmente si on tape la balle (raquette
            gauche)
            if(getBall().position.x >= getRacket()[i].position.x +
                getRacket()[i].width)
                this.getScore()[0]++;
        }
    }
}

getBall().position.translate(getBall().speed.x, getBall().speed.y);
if (getBall().position.x < 0){ //Si la balle tape le bord gauche
    de l'ecran
    try {
        //Mode multijoueur
        if(nbPlayer > 1){
            this.getScore()[1]++; //le score du joueur droit augmente
            for(int i = 0; i < 2; i++){
                if(getRacket()[i].bonus > -1){
                    //Si un bonus existe, on augmente de 1 sa variable
                    bonus (disparait apres 2 tours)
                    getRacket()[i].updateBonus();
                }
            }
        }
        else{
            this.getScore()[0] = 0; //En mode solo, le score du joueur
            repart a 0
        }
        this.getBall().setLastPositionY(this.getBall().position.y);
        this.getBall().setLastTouch(2);
        Thread.sleep(1000);
        getBall().position.x = 500;
        getBall().position.y = 300;
        getBall().speed.x = -getBall().BALL_SPEED;
        getBall().speed.y = -getBall().BALL_SPEED;
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

//Rebond sur le bord haut de l'ecran
if (getBall().position.y < 0){
    getBall().position.y = 0;
    getBall().speed.y = -getBall().speed.y;
}
//Si la balle tape le bord droit de l'ecran
if (getBall().position.x > SIZE_PONG_X){

```

```

try {
    if (nbPlayer > 1){ //On fait la meme chose que pour le bord
        gauche
        this.getScore()[0]++;
        for(int i = 0; i < 2; i++){
            if(getRacket()[i].bonus > -1){
                getRacket()[i].updateBonus();
            }
        }
        this.getBall().setLastTouch(1);
        this.getBall().setLastPositionY(this.getBall().position.y);
    }
    Thread.sleep(1000);
    getBall().position.x = 500;
    getBall().position.y = 300;
    getBall().speed.x = getBall().BALL_SPEED;
    getBall().speed.y = getBall().BALL_SPEED;
} catch (InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}

if (getBall().position.y > SIZE_PONG_Y - getBall().height){
    //Rebond sur le bas de l'ecran
    getBall().position.y = SIZE_PONG_Y - getBall().height;
    getBall().speed.y = -getBall().speed.y;
}

if(this.nbPlayer > 1){//Multi
    animateBallBonus(); //On anime les bonus
}

/* Et on met a jour l'ecran */
updateScreen();
}

public void animateBallBonus(){
    for(int j = 0; j < nbBonus; j++){ //Pour les bonus du tableau
        tabBallBonus

        if(this.tabBallBonus[j].getInstance()){//S'il existe on le fait
            avancer
            this.tabBallBonus[j].position.translate(this.tabBallBonus[j].getBonusSpeed(),
                0);
        }

        if( this.tabBallBonus[j].apparition(this.getScore()[0],
            this.getScore()[1]) && !this.tabBallBonus[j].getInstance()
            && !this.tabBallBonus[j].isHist(this.getScore()[0],
            this.getScore()[1])){

```



```

        this.tabBallBonus[j].setInstance(true); //Initialisation de
        la balle, si condition d'apparition (fonction) et si il
        n'en existe pas deja un et si le score n'as pas deja
        creer une balle
        this.tabBallBonus[j].position.x = 500;
        this.tabBallBonus[j].position.y = getBall().lastPositionY +
        this.tabBallBonus[j].getPositionY();
        this.tabBallBonus[j].setHist(this.getScore()[0],
        this.getScore()[1]);
        if(getBall().getLastTouch() == 1) //Definition de
        l'orientation de la ballBonus
            this.tabBallBonus[j].setBonusSpeedLeft();
        else
            this.tabBallBonus[j].setBonusSpeedRight();
    }

    for(int i = 0; i < 2; i++){
        if(collision(this.tabBallBonus[j], getRacket()[i]) &&
        this.tabBallBonus[j].getInstance()){
            this.tabBallBonus[j].setInstance(false);
            this.tabBallBonus[j].deformationRacket(this.getRacket()[i]);
            this.getRacket()[i].bonus = 0;
        }
    }

    for(int i = 0; i < 2; i++){
        if(getRacket()[i].bonus == 2){//Au bout de deux tours, les
        bonus disparaissent
            getRacket()[i].bonus = -1;
            getRacket()[i].setImage("image/racket.png");
            if(getRacket()[0].position.x == 50)
                this.RACKET_SPEED_JG = 15;
            else
                this.RACKET_SPEED_JD = 15;
        }
    }
}

public int getPosOtherRacket(){ //Donne un chiffre aleatoire,
correspondant a une raquette coupe en 5
    int lower = 1;
    int higher = 6;
    int random = (int)(Math.random() * (higher-lower)) + lower;
    return getRacket()[1].height/random;
}

public boolean collision(Ball ball, Racket racket){
    if((racket.position.x >= ball.position.x + ball.width)// trop a
    droite

```

```

    || (racket.position.x + racket.width <= ball.position.x) // trop
        a gauche
    || (racket.position.y >= ball.position.y + ball.height) // trop
        en bas
    || (racket.position.y + racket.height <= ball.position.y)) //
        trop en haut
        return false;
else
    return true;
}

public void rebound(int i){
    if((ball.position.x + ball.width <= racket[i].position.x) ||
        (ball.position.x >= racket[i].position.x + racket[i].width) ){

        int sp = this.getBall().position.y + this.getBall().height/2;
        int rk = this.getRacket()[i].position.y +
            this.getRacket()[i].height/5;

        if(sp <= rk){
            getBall().speed.y = getBall().speed.y - 1;
        }
        else{
            if(sp <= 2*rk){
                getBall().speed.y = getBall().speed.y -1/2;
            }
            else{
                if(sp <= 4*rk){
                    getBall().speed.y = getBall().speed.y +1/2;
                }
                else{
                    if(sp <= 5*rk){
                        getBall().speed.y = getBall().speed.y +1;
                    }
                }
            }
        }
        ball.speed.x = - ball.speed.x;
    }
    else{
        if(Math.signum(racket[i].speed) != Math.signum(ball.speed.y)){
            ball.speed.y = -ball.speed.y;
        }
        else{
            ball.position.y += ball.speed.y;
        }
    }
}
}

```

```

/*Controle quand le joueur presse un bouton*/
public void keyPressed(KeyEvent e) {
    switch(e.getKeyCode()) {
        case KeyEvent.VK_UP:
        case KeyEvent.VK_KP_UP:
            if(getRacket()[0].position.x == 50)
                getRacket()[0].speed = -RACKET_SPEED_JG;
            else
                getRacket()[0].speed = -RACKET_SPEED_JD;
            break;
        case KeyEvent.VK_DOWN:
        case KeyEvent.VK_KP_DOWN:
            if(getRacket()[0].position.x == 50)
                getRacket()[0].speed = RACKET_SPEED_JG;
            else
                getRacket()[0].speed = RACKET_SPEED_JD;
            break;
        default:
            System.out.println("got press "+e);
    }
}

/*Controle quand le joueur relache un bouton*/
public void keyReleased(KeyEvent e) {
    switch (e.getKeyCode()) {
        case KeyEvent.VK_UP:
        case KeyEvent.VK_KP_UP:
            getRacket()[0].speed = 0;
            break;
        case KeyEvent.VK_DOWN:
        case KeyEvent.VK_KP_DOWN:
            getRacket()[0].speed = 0;
            break;
        default:
            System.out.println("got release "+e);
    }
}

public void keyTyped(KeyEvent e) { }

/*
 * (non-Javadoc) This method is called by the AWT Engine to paint what
 * appears in the screen. The AWT engine calls the paint method every
 * time
 * the operative system reports that the canvas has to be painted.
 * When the
 * window is created for the first time paint is called. The paint
 * method is
 * also called if we minimize and after we maximize the window and if
 * we

```

```

    * change the size of the window with the mouse.
    *
    * @see javax.swing.JComponent#paint(java.awt.Graphics)
    */

public void paint(Graphics g) {
    g.drawImage(buffer, 0, 0, this);
}

/**
 * Draw each Pong item based on new positions
 */
public void updateScreen() {
    if (buffer == null) {
        /*Au premier appel on initialise window*/
        buffer = createImage(SIZE_PONG_X, SIZE_PONG_Y);
        if (buffer == null)
            throw new RuntimeException("Could not instanciate graphics");
        else
            graphicContext = buffer.getGraphics();
    }
    /* Colorie la fond avec la couleur backgroundColor, ici noir*/
    graphicContext.setColor(backgroundColor);
    graphicContext.fillRect(0, 0, SIZE_PONG_X, SIZE_PONG_Y);
    /*dimension de la fenetre*/

    /* Dessine les different objets*/
    graphicContext.drawImage(getBall().image, getBall().position.x,
        getBall().position.y, getBall().width, getBall().height, null);
    for(int i = 0; i < 2; i++){
        graphicContext.drawImage(getRacket()[i].image,
            getRacket()[i].position.x, getRacket()[i].position.y,
            getRacket()[i].width, getRacket()[i].height, null);
    }

    for(int i = 0; i < 6; i++){
        graphicContext.drawImage(net.image, 500, 1*(i*106), null);
    }

    for(int j = 0; j < nbBonus; j++){
        if(this.tabBallBonus[j].getInstance()){
            graphicContext.drawImage(this.tabBallBonus[j].image,
                this.tabBallBonus[j].position.x,
                this.tabBallBonus[j].position.y,
                this.tabBallBonus[j].width, this.tabBallBonus[j].height,
                null);
        }
    }

    Font font = new Font("Courier", Font.BOLD, 100);

```

```

        graphicContext.setFont(font);
        graphicContext.setColor(Color.white);
        String scoreLeft = Integer.toString(this.getScore()[0]);
        if(this.nbPlayer > 1){
            String scoreRight = Integer.toString(this.getScore()[1]);
            graphicContext.drawString(scoreRight, 550, 100);
        }
        graphicContext.drawString(scoreLeft, 400, 100);
        this.repaint();
    }

    public Racket[] getRacket() {
        return racket;
    }

    public void setRacket(Racket[] racket) {
        this.racket = racket;
    }

    public Ball getBall() {
        return ball;
    }

    public void setBall(Ball ball) {
        this.ball = ball;
    }

    public static int getSizePongY(){
        return SIZE_PONG_Y;
    }

    public static int getSizePongX(){
        return SIZE_PONG_X;
    }
}

```

7.13 PongItem

```
package pong.gui;

import java.awt.Point;

public class PongItem {
    protected int width;
    protected int height;
    protected Point position;
}
```

7.14 Protected

```
package pong.gui;

import pong.Main;

public class Protected {

    public int ancientPositionBall = 0;
    public int ancientPositionRacket = 0;

    public void verificationRacket(Pong pong){
        if(pong.getRacket()[0].position.x == 50){ //On differencie la
            raquette droite de la raquette gauche
            if(pong.getRacket()[0].bonus == 0 && pong.getRacket()[1].bonus
                == 0){ //Si les bonus ne sont pas active
                if (((int) Main.otherRacket.getY() !=
                    this.ancientPositionRacket + Pong.RACKET_SPEED_JG &&
                    (int) Main.otherRacket.getY() !=
                    this.ancientPositionRacket - Pong.RACKET_SPEED_JG) &&
                    (int) Main.otherRacket.getY() !=
                    this.ancientPositionRacket){
                    System.out.println("Il y a triche"); //Et que la position
                        recue pour la raquette adverse ne correspond pas a ce
                        que l'on attendais, il y a triche
                }
            }
            this.ancientPositionRacket = Main.otherRacket.y; //Sinon
                l'ancienne position de la raquette adverse et retenue pour
                le calcul suivant
        }
        else{//Meme chose si on est a droite dans le jeu
            if(pong.getRacket()[0].bonus == 0 && pong.getRacket()[1].bonus
                == 0){
                if (((int) Main.otherRacket.getY() !=
                    this.ancientPositionRacket + Pong.RACKET_SPEED_JD &&
                    (int) Main.otherRacket.getY() !=
                    this.ancientPositionRacket - Pong.RACKET_SPEED_JD) &&
                    (int) Main.otherRacket.getY() !=
                    this.ancientPositionRacket){
                    System.out.println("Il y a triche");
                }
            }
            this.ancientPositionRacket = Main.otherRacket.y;
        }
    }
}
```

7.15 Racket

```
package pong.gui;

import java.awt.Image;
import java.awt.Point;
import java.awt.Toolkit;
import javax.swing.ImageIcon;

public class Racket extends PongItem{

    protected Image image;
    protected int speed;
    protected int bonus = -1;

    public Racket(int joueur){
        this.setImage("image/racket.png");
        if(joueur == 1){
            this.position = new Point(50, 0);
        }
        //Mode solo ou 2 joueurs
        if (joueur == 0 || joueur == 2){
            this.position = new Point(930 ,0);
        }
    }

    public Racket(int x, int y){
        this.position = new Point(x, y);
    }

    public void setImage(String im){
        ImageIcon icon;
        this.image = Toolkit.getDefaultToolkit().createImage(
            ClassLoader.getResource(im));
        icon = new ImageIcon(image);
        this.width = icon.getIconWidth();
        this.height = icon.getIconHeight();
    }

    public int getX(){
        return this.position.x;
    }

    public int getY(){
        return this.position.y;
    }
}
```



```
public int getBonus(){
    return this.bonus;
}

public void setBonus(int i){
    this.bonus = i;
}

public void updateBonus(){
    this.bonus++;
}

public int getSpeed(){
    return this.speed;
}

public void setSpeed(int i){
    this.speed = i;
}

public Racket NextPosition(){
    return new Racket(this.position.x, this.position.y+this.speed);
}
}
```

7.16 Réseau

```
package pong.gui;

import java.awt.Point;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

import pong.Main;

public class Reseau{

    /*
     * Envoyer une coordonnee
     */
    public void send(Socket sock, Point point, String coor){
        ObjectOutputStream outToServer;
        double pos;
        try {
            outToServer = new ObjectOutputStream(sock.getOutputStream());
            //On ecrit la coordonnee String
            outToServer.writeObject(coor);
            //S'il s'agit de la coordonnee x du point courant
            if(coor.equals("x_racket") || coor.equals("x_ballpos") ||
                coor.equals("x_ballspeed"))
                pos = point.getX();
            else
                pos = point.getY();
            //On ecrit cette coordonnee
            outToServer.writeDouble(pos);
            //On envoie le tout vers le reseau
            outToServer.flush();
        }
        catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /*
     * Recevoir une coordonnee
     */
    public void read(Socket sock){
        ObjectInputStream inServer;
        ObjectOutputStream outToServer;
        String coor = null;
        //Lecture
```

```

try {
    inServer = new ObjectInputStream(sock.getInputStream());
    outToServer = new ObjectOutputStream(sock.getOutputStream());
    try {
        //Cast la coordonnee d'Object en String
        coor = (String) inServer.readObject();
        /*
         * Selon la valeur de coor, on recupere ecrit la coordonnee
         * suivante dans le reseau a l'emplacement correspondant
         * puis on envoie un ack sous forme de String
         */
        if(coor.equals("x_racket")){
            Main.otherRacket.setLocation(inServer.readDouble(),
                Main.otherRacket.getY());
            outToServer.writeObject("x_racket_recu");
        }
        else{
            if(coor.equals("y_racket")){
                Main.otherRacket.setLocation(Main.otherRacket.getX(),
                    inServer.readDouble());
                outToServer.writeObject("y_racket_recu");
            }
            else{
                if(coor.equals("x_ballpos")){
                    Main.ballPosition.setLocation(inServer.readDouble(),
                        Main.ballPosition.getY());
                    outToServer.writeObject("x_ballpos_recu");
                }
                else{
                    if(coor.equals("y_ballpos")){
                        Main.ballPosition.setLocation(Main.ballPosition.getX(),
                            inServer.readDouble());
                        outToServer.writeObject("y_ballpos_recu");
                    }
                    else{
                        if(coor.equals("x_ballspeed")){
                            Main.ballSpeed.setLocation(inServer.readDouble(),
                                Main.ballSpeed.getY());
                            outToServer.writeObject("x_ballspeed_recu");
                        }
                        else{
                            if(coor.equals("y_ballspeed")){
                                Main.ballSpeed.setLocation(Main.ballSpeed.getX(),
                                    inServer.readDouble());
                                outToServer.writeObject("y_ballspeed_recu");
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
} catch (ClassNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}

/*
 * meme principe que send mais avec un int
 */
public void sendMaxScore(Socket sock, int scoreMax){
    ObjectOutputStream outToServer;
    try {
        outToServer = new ObjectOutputStream(sock.getOutputStream());
        outToServer.writeObject("score");
        outToServer.writeInt(scoreMax);
        outToServer.flush();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

/*
 * meme principe que read, avec un int
 */
public void readMaxScore(Socket sock){
    ObjectInputStream inServer;
    ObjectOutputStream outToServer;
    try {
        inServer = new ObjectInputStream(sock.getInputStream());
        outToServer = new ObjectOutputStream(sock.getOutputStream());
        String response;
        try {
            response = (String) inServer.readObject();
            if(response.equals("score")){
                Main.setScoreMax(inServer.readInt());
                outToServer.writeObject("score_recu");
                outToServer.flush();
            }
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    } catch (IOException e) {
        // TODO Auto-generated catch block

```

```

        e.printStackTrace();
    }
}

/*
 * Traitement d'un ack
 */
public boolean ack(Socket sock, String message){
    ObjectInputStream inServer;
    String response;
    try {
        inServer = new ObjectInputStream(sock.getInputStream());
        try {
            //On recupere le String souhaite
            response = (String) inServer.readObject();
            //Si la reponse correspond au message attendu
            if(response.equals(message))
                return true;
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    //sinon
    return false;
}
}

```

7.17 Tests

```
package pong.gui;

public class Tests {

    Pong pong;
    Window window;

    public Tests(){
        pong = new Pong(1, 2);
        window = new Window(pong);
    }

    public int TestsCollision(){
        this.InitialisationPosition();
        //Parcours des 2 raquettes
        for(int i = 0; i < 2; i++){
            for(int y_racket = 0; y_racket < Pong.getSizePongY() -
                this.pong.getRacket()[0].height; y_racket++){
                //Positionnement de la racket courante en y
                this.pong.getRacket()[i].position.setLocation(0, y_racket);
                for(int x_ball = 0; x_ball < Pong.getSizePongX() -
                    this.pong.getBall().width; x_ball++){
                    for(int y_ball = 0; y_ball < Pong.getSizePongY() -
                        this.pong.getBall().height; y_ball++){
                        //Positionnement de la balle
                        this.pong.getBall().position.setLocation(x_ball,
                            y_ball);
                        /*
                         * Si la balle n'est pas censée collisionner, selon nos
                         * valeurs est que collision
                         * retourne true (ou l'inverse), alors il y a une erreur
                         */
                        if(((this.pong.getRacket()[i].getX() >= x_ball +
                            this.pong.getBall().width)
                            || (this.pong.getRacket()[i].getX() +
                                this.pong.getRacket()[i].width <= x_ball)
                            || (y_racket >= y_ball +
                                this.pong.getBall().height)
                            || (y_racket + this.pong.getRacket()[i].height <=
                                y_ball))
                            == (this.pong.collision(this.pong.getBall(),
                                this.pong.getRacket()[i]))){
                            System.out.println("ERREUR TEST COLLISION");
                            return -1;
                        }
                    }
                }
            }
        }
    }
}
```

```

    }
}
}
//Si le test s'est bien termine
return 0;
}

public int TestsRebound(){
    this.InitialisationPosition();
    int lastSpeedY = Ball.BALL_SPEED;
    int lastSpeedX = Ball.BALL_SPEED;
    int lastPositionY = pong.getBall().position.y;
    pong.getRacket()[0].speed = Pong.RACKET_SPEED_JG;
    pong.getRacket()[1].speed = Pong.RACKET_SPEED_JD;
    //Parcours des raquettes
    for(int i = 0; i < 2; i++){
        for(int y_racket = 0; y_racket < Pong.getSizePongY() -
            this.pong.getRacket()[0].height; y_racket++){
            //Deplacement raquette courante
            this.pong.getRacket()[i].position.setLocation(0, y_racket);
            for(int x_ball = 0; x_ball < Pong.getSizePongX() -
                this.pong.getBall().width; x_ball++){
                for(int y_ball = 0; y_ball < Pong.getSizePongY() -
                    this.pong.getBall().height; y_ball++){
                    //Deplacement balle courante
                    this.pong.getBall().position.setLocation(x_ball,
                        y_ball);
                    //La fonction collision a ete verifiee precedemment et
                    est donc utilisable
                    if(pong.collition(this.pong.getBall(),
                        this.pong.getRacket()[i])){
                        lastSpeedY = this.pong.getBall().speed.y;
                        lastSpeedX = this.pong.getBall().speed.x;
                        lastPositionY = this.pong.getBall().position.y;
                        this.pong.rebound(i);
                        if((x_ball + this.pong.getBall().width <=
                            this.pong.getRacket()[i].position.x ||
                            (x_ball >= this.pong.getRacket()[i].position.x
                                + this.pong.getRacket()[i].width) ){
                            int sp = this.pong.getBall().position.y +
                                this.pong.getBall().height/2;
                            int rk = this.pong.getRacket()[i].position.y +
                                this.pong.getRacket()[i].height/5;
                            if(sp <= rk){
                                if(this.pong.getBall().speed.y != lastSpeedY -
                                    1 && this.pong.getBall().speed.x != -
                                    lastSpeedX){
                                    return -1;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```



```
        return 0;
    }

    public void InitialisationPosition(){
        //Initialisation de la position des raquettes
        this.pong.getRacket()[0].position.setLocation(50, 0);
        this.pong.getRacket()[1].position.setLocation(930, 0);
        //Initialisation de la position de la balle
        this.pong.getBall().position.setLocation(0, 0);
    }
}
```

7.18 Window

```
package pong.gui;

import java.awt.Point;
import java.net.Socket;
import javax.swing.JFrame;

import pong.Main;

/**
 * A Window is a Java frame containing an Pong
 */
public class Window extends JFrame {

    private static final long serialVersionUID = 1L;

    /**
     * Pong component to be displayed
     */
    private final Pong pong;

    /**
     * Constructor
     */
    public Window(Pong pong) {
        this.pong = pong;
        this.addKeyListener(pong);
    }

    /**
     * Displays the Window using the defined margins, and call the
     * {@link Pong#animate()} method of the {@link Pong} every 100ms
     */
    public void displayOnscreen(Socket sock) {
        int fin = -1; // Si la variable passe 1, un joueur a quitte la partie
        add(pong);
        pack();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // On ferme la
            fenetre si on appuye sur la croix
        setVisible(true);
        Reseau r = new Reseau();
        Protected p = new Protected();
        // Gestion du score max
        if (Main.getIndiceSC() == 0) { // Serveur
            do {
                r.sendMaxScore(sock, Main.getScoreMax());
            } while (!r.ack(sock, "score_recu"));
        }
    }
}
```

```

else{//Client
    r.readMaxScore(sock);
}
while (true){
    pong.animate();
    if(pong.getScore()[0] == Main.getScoreMax() ||
        pong.getScore()[1] == Main.getScoreMax()){ //Si un des
        joueurs a atteint le score max
        BoxGameOver boxGameOver = new BoxGameOver(); //On affiche la
        boite de dialogue BoxGameOver
        if(pong.getScore()[0] == Main.getScoreMax()){
            fin = boxGameOver.BoxGameOver(pong, this, 1); //On met la
            reponse des joueurs dans fin
        }
        if(pong.getScore()[1] == Main.getScoreMax()){
            fin = boxGameOver.BoxGameOver(pong, this, 2);
        }
    }
    try {
        Thread.sleep(10);
    } catch (InterruptedException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    if(pong.ballMyTurn()){
        Point[] ballData = new Point[2];
        ballData[0] = pong.getBallPosition();
        ballData[1] = pong.getSpeedPosition();
        //Envoi coordonnees ball, position et vitesse (x puis y) avec
        demande d'ack
        do{
            r.send(sock, ballData[0], "x_ballpos");
        }while(!r.ack(sock, "x_ballpos_recu"));
        do{
            r.send(sock, ballData[0], "y_ballpos");
        }while(!r.ack(sock, "y_ballpos_recu"));
        do{
            r.send(sock, ballData[1], "x_ballspeed");
        }while(!r.ack(sock, "x_ballspeed_recu"));
        do{
            r.send(sock, ballData[1], "y_ballspeed");
        }while(!r.ack(sock, "y_ballspeed_recu"));
    }
    else{
        //1 read par donnee envoyee
        r.read(sock);
        r.read(sock);
        r.read(sock);
        r.read(sock);
    }
}

```

```

        //Envoi coordonnees de la racket
        do{
            r.send(sock, pong.getCurrentRacketPosition(), "x_racket");
            r.read(sock);
        }while(!r.ack(sock, "x_racket_recu"));

        do{
            r.send(sock, pong.getCurrentRacketPosition(), "y_racket");
            r.read(sock);
        }while(!r.ack(sock, "y_racket_recu"));

        p.verifcationRacket(pong);
        try {
            Thread.sleep(Pong.timestep);
        } catch (InterruptedException e) {
            // Rien
        };
        if(fin == 1){
            break;
        }
    }
}

public void displayOnscreen(){//Mode solo, on enleve la partie
    reseau, et le score max de la fonction precedente
    add(pong);
    pack();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
    while (true){
        pong.animate();
        try {
            Thread.sleep(10);
        } catch (InterruptedException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
    }
}
}

```
