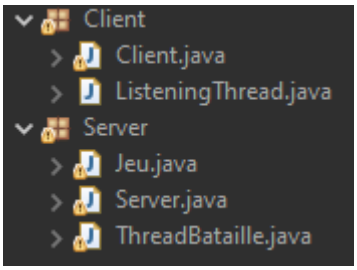
MEDNINI
Soufyen
IA2R – TD8

TP Socket : Bataille Navale

(voir à la fin pour mes notes sur le projet)

Ce projet permet à deux clients, liés à un serveur, de réaliser une fameuse Bataille Navale.

Le projet se décompose en deux packages (client et serveur) :



Toute la partie « jeu » est placé du coté serveur, le client a juste à se connecter par l'intermédiaire d'un thread.

- *Classe « Serveur »*

```java
package Server;

import java.net.ServerSocket;

public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket ecoute = new ServerSocket(1234);


            System.out.println("Serveur lancé!");
            int id=0;

            while(true) {
                Socket client1 = ecoute.accept();
                Socket client2 = ecoute.accept();
                new ThreadBataille(id,client1,client2).start();
                id++;
                }

            }


        catch(Exception e) {e.getMessage();}{
            // traitement d'erreur
        }
    }

}
```

Cette classe va générer un nouveau socket et attendre que deux clients uniquement soient connectés et soient en « écoute » de ce socket, ce qui va permettre de lancer le thread coté client et ainsi pouvoir commencer la partie. Elle est écrite de sorte à ce que 2 clients, ni moins ni plus, puissent se battre. Deux clients autres peuvent également procéder à une bataille en simultanée, cela créera un nouveau thread.

- *Classe « Jeu »*

La classe est le cœur même de la bataille. Elle contient les méthodes principales pour les deux joueurs.

```
int[][] grille;
int count = 0;

public Jeu() {
    grille = new int[10][10];
    for(int j=0;j< grille.length;j++) {
        for(int i=0;i<grille.length;i++) {
            grille[i][j] = 0;
        }
    }


}
```

Constructeur de la classe permettant de créer une grille. On remarque la définition d'un entier « count » qui nous servira à un test pour la fin de partie.

Méthode Afficher :

```
public String Afficher() {
    String grid = "---> \nVotre grille : \n"
                  + "\n    0 1 2 3 4 5 6 7 8 9"
                  + "\n    --------------------\n";

    for(int j=0;j< grille.length;j++) {
        grid += j + " | ";
        for(int i=0;i<grille.length;i++) {


            grid += grille[i][j] + " ";
        }
        grid += "\n";
    }

    return grid;

}
```

Elle affichera la grille personnelle sur chaque client. Avec numérotation des lignes et des colonnes de 0 à 9.

Méthode testPos :

```
public boolean testPos(int lin, int col) {
    if (grille[lin][col] == 1) {
        return true;
    }
    else {
        return false;
    }

}
```

Cette méthode permettra, pour une coordonnée en entrée, de tester si la case est égale à 1 ou non. Elle va être utile dans les méthodes d'attaques et de placement de bateau.

Méthode creerBateau :

J'ai choisi de faire cette méthode en prenant en paramètre 4 coordonnées du client et d'effectuer plusieurs tests pour savoir si elles sont correctes. A la suite de ces tests, un booléen est renvoyé pour savoir si le bateau a pu être placé ou non.

```
public boolean creerBateau(int lin1, int lin2, int col1, int col2) {
    int valLin = Math.abs(lin1-lin2);
    int valCol = Math.abs(col1-col2);
    boolean batcr= false;

    if(lin1<=9 && lin2<=9 && col1<=9 && col2<=9) {


    if (lin1 == lin2 && col1 != col2 && valCol<5) {
        if(col1<col2) {
            for(int i =col1;i<col2+1;i++) {
                if(testPos(i-1,lin1-1)==false) {
                    grille[i-1][lin1-1] = 1;
                    batcr = true;
                }
                else {
                    System.out.println("Il y a déjà un bateau ici : Ligne " + lin1);
                    batcr = false;

                }
            }
        }
        else {
            for(int i = col2;i<col1+1;i++) {
                if(testPos(i-1,lin1-1)==false) {
                    grille[i-1][lin1-1] = 1;
                    batcr = true;
                }
                else {
                    System.out.println("Il y a déjà un bateau ici : Ligne " + lin1);
                    batcr = false;
                }

            }
        }
    }
```

Ici nous avons une première condition qui vérifie que les coordonnées entrées par le client sont bien comprises entre 0 et 9. Également le client peut entrer ses coordonnées dans n'importe quel sens, cela fonctionnera (exemple : lin 1=6 et lin 2=4 reviendra à lin 1=4 et lin 2=6).

Puis les tests, avec vérification de la taille du bateau (<5) par valeur absolue, sont dans l'ordre :

- Mêmes lignes mais différentes colonnes
- Différentes lignes mais mêmes colonnes
- Mêmes lignes et mêmes colonnes (bateau de 1)

Cette méthode test également s'il y a déjà un bateau sur les coordonnées entrées, dans lequel cas il renvoie un message notifiant ceci au client.

Pour un bateau sur la ligne 3 de la colonne 2 à 4, le client devra entrer 3,3,2,4.

Enfin, si à la suite de ces tests, le retour du booléen est "false", alors le bateau n'a pas été placé et on proposera à l'utilisateur de recommencer.

Méthode attack :

```
public boolean attack(int l, int c) {

    if(grille[c][l] == 1) {
        return true;
    }
    else {
        return false;
    }
}
```

Pour un x et un y, cette méthode retournera « true » si les coordonnées de l'attaque correspondent à des coordonnées existantes.

Méthode fin :

```
public int fin() {
        for(int i = 0; i<grille.length; i++){
            for(int j = 0; j<grille[i].length; j++){
                if(testPos(i,j)==true){
                    count++;
                }
            }
        }

    return count;
}
```

Enfin cette méthode va permettra, à chaque fois que le test de position retourne « true », d'incrémenter une variable count qui permettra plus tard de connaitre la fin de la partie.

- *Classe « ThreadBataille » :*

```
public class ThreadBataille extends Thread{
int id;
BufferedReader in1;
PrintWriter out1;
BufferedReader in2;
PrintWriter out2;
Jeu gr1;
Jeu gr2;
String p1;
String p2;
```

Cette classe, qui a pour classe mère Thread, va permettre aux deux clients de communiquer avec le thread et le jeu. Pour chaque client une grille.

```
public ThreadBataille(int id,Socket client1,Socket client2) {

    try {
        this.id=id;

        in1 = new BufferedReader(new InputStreamReader(client1.getInputStream()));
        out1 = new PrintWriter(client1.getOutputStream(), true);

        in2 = new BufferedReader(new InputStreamReader(client2.getInputStream()));
        out2 = new PrintWriter(client2.getOutputStream(), true);




    }

    catch (Exception e) {}
}
```

Constructeur qui permettra de lire et d'écrire coté client par l'intermédiaire de BufferedReader et PrintWriter.

Méthode addBateauJ1 (le client 2 dispose de la même méthode addBateauJ2) :

```
public String addBateauJ1() {
    try {
        gr1 = new Jeu();
        int i =0;
        int a1;
        int b1;
        int c1;
        int d1;
        while(i!=3) {

            out1.println("Il vous reste " + (5-i) + " bateaux à placer!" );
            out1.println("Entrer les coordonées d'un bateau (0-9):");

                out1.println("Ligne 1:");
                a1  = Integer.parseInt(in1.readLine());


                while (a1 >= 9) {
                    out1.println("Ligne 1:");
                    a1  = Integer.parseInt(in1.readLine());
                }

                out1.println("Ligne 2:");
                b1 = Integer.parseInt(in1.readLine());

                while (b1 >=9) {
                    out1.println("Ligne 2:");
                    b1 = Integer.parseInt(in1.readLine());
                }

                out1.println("Colonne 3:");
                c1 = Integer.parseInt(in1.readLine());
                while(c1 >=9) {
                    out1.println("Colonne 3:");
                    c1 = Integer.parseInt(in1.readLine());
                }

                out1.println("Colonne 4:");
                d1 = Integer.parseInt(in1.readLine());
                while(d1 >=9) {
                    out1.println("Colonne 4:");
                    d1 = Integer.parseInt(in1.readLine());
                }
```

Tout d'abord on demande au client d'entrée des coordonnées pour placer ses bateau (cf. méthode créerBateau). On vérifie que le client entre bien un chiffre entre 0 à 9.

```
            if(gr1.creerBateau(a1, b1, c1, d1) == true) {

                out1.println("Bateau créé!");
                i++;
            }
            else {
                if(gr1.testPos(a1-1, c1-1) == true || gr1.testPos(b1-1, d1-1) == true) {
                    out1.println("Il y a déjà un bateau ici!");
                }
                else {
                    out1.println("Coordonnées incorrectes : Un bateau fait maximum 5 cases !\n"
                            + "Vous devez inscrire la ligne1, ligne2, colonne1, colonne2\n"
                            + "Exemple (1,1,2,3) pour un bateau sur la ligne 1 de la colonne 2 à 3"
                            + "\n------------------------------------------------------------");
                }

            }

        }


        String grid1 = gr1.Afficher();
        return grid1;

    }
    catch (Exception e) {
        // TODO: handle exception
    }
    return null;
}
```

Par la suite on vérifie donc si la méthode créerBateau renvoie « true », si c'est le cas, le client a donc son bateau dans sa grille et on incrémente la variable i de 1 correspondant au nombre de bateau, le client a le droit de poser 5 bateaux. Également, on test si les coordonnées du bateau ne correspondent pas à un bateau existant mais aussi si c'est parce qu'il y a déjà un bateau ou parce que le client a mal entré les coordonnées par l'intermédiaire de testPos.

Méthode attackJ1 (le client 2 dispose de la même méthode attackJ2) :

```
public boolean attackJ1() {
    try {
        out2.println(p1 +" vous attaque!");
        out1.println("Veuillez entrez des coordonnées pour attaquez " + p2 + " :");
        out1.println("X :"); int x = Integer.parseInt(in1.readLine());
        out1.println("Y :"); int y = Integer.parseInt(in1.readLine());
        if (gr2.attack(x, y)==true) {
            out1.println("Touché");
            return true;
        }
        else {
            out1.println("Non touché");
            return false;
        }
    } catch (Exception e) {
        // TODO: handle exception
    }
    return false;
}
```

Ici on informe le client 2 qu'il est attaqué par le client 1, qui entre des coordonnées (un x et un y) puis on fait appel à la méthode attack qui nous retourne si nous avons touché sur la grille du client 2 (true si dans la grille = 1)

Méthode finDePartie :

```
public boolean finDePartie() {
    int fin1 = gr1.fin();
    int touch1 = 0;

    int fin2= gr2.fin();
    int touch2 = 0;

    while (fin1 != touch1 && fin2 != touch2) {
        if (attackJ1()==true) {
            out1.println("Vous avez touché " + (touch2+1) + " fois " + p2);
            out2.println(p1 + " vous a touché " + (touch2+1) + " fois");
            touch2++;
        }
        else {

            out2.println(p1 +" vous a raté");
        }


        if (attackJ2()==true) {
            out2.println("Vous avez touché " + (touch1+1) + " fois " +p1);
            out1.println(p2 + " vous a touché " + (touch1+1) + " fois");
            touch1++;
        }
        else {
            out1.println(p2 + " vous a raté");
        }

        System.out.println(fin1 + " " + touch1);
        System.out.println(fin2 + " " + touch2);

    }


    if(fin1==touch1) {
        return true;
    }
    else {
        return false;
    }
```

Par l'intermédiaire d'un compteur, on vérifie que le client 1 a touché, ou non, tous les bateaux du client 2. Une fois que le nombre de touche est égale au nombre de 1 dans la grille adverse (cf. méthode fin), le client correspondant gagne, et la méthode renvoie true.

Méthode run :

```
out1.println("--> Joueur 1, saississez votre nom : <--");
out2.println("--> Le joueur 1 est entrain de saisir son nom. <--");
p1 = in1.readLine();
out1.println("--> Le joueur 2 est entrain de saisir son nom. <--");
out2.println("--> Joueur 2, saississez votre nom : <--");
p2 = in2.readLine();



out1.println("--> " + p1 + " doit placer ses bateaux ! <--");
out2.println("--> " + p1 + " doit placer ses bateaux ! <--");
out1.println(addBateauJ1());

out1.println("--> " + p2 + " doit placer ses bateaux ! <--");
out2.println("--> " + p2 + " doit placer ses bateaux ! <--");
out2.println(addBateauJ2());

if(finDePartie()==true) {
    out1.println("\n--- "+ p2 + " gagne ! ---");
    out2.println("\n--- " + p2 +" gagne ! ---");

}
else {

    out1.println("\n--- " + p1 +" gagne ! ---");
    out2.println("\n--- "+ p1 +" gagne ! ---");
}
```

Enfin cette dernière méthode va donc donner des indications aux deux clients, faire appel à la méthode d'ajout des bateaux et leur indiquer qui des deux clients a gagné la partie.

- *Classe Client et ListeningThread :*

La classe Client va permettre la connexion au thread et d'afficher tous les messages envoyés par la classe ListeningBataille.

Le client peut mettre un terme à la partie en tappant simple « fin » dans sa console.

La classe ListeningThread est destiné au client afin justement d'écouter le thread.

Notes :

Le projet a été amené à terme. Les clients peuvent placés 5 bateaux sur leurs grilles et essayer de détruire les bateaux adverses. Tout fonctionne parfaitement et deux parties peuvent être déroulées en simultanées.

Toutes les données du jeu sont du coté serveur. Les clients mémorisent toutes leurs informations.

J'ai eu quelques soucis pour faire une méthode « couler » pour assigner une série de point à un bateau, ne voyant pas comment faire, j'ai omis cette partie, mais touy ce qu'on attend d'une bataille navale est présent dans ce programme.

Également j'ai choisi d'entrer les coordonnées d'un bateau sous cette forme, qui me paraissait la meilleure méthode mais sachant pertinemment qu'il y a toujours mieux. Ayant discuté avec mes camarades j'ai compris comment j'aurais pu concevoir d'une autre façon cette méthode. Néanmoins elle reste fonctionnelle et c'est ce qu'on attend ici.