



TER FMIN200

—

Développement d'un jeu de type Bomberman en réseau  
sous Android et iOS

BONVILA Olivier

COUSEIN Kilian  
TARDIEU Benjamin

PITOT Ludovic

Nous tenons à remercier M. Giroudeau qui a toléré notre fâcheuse tendance à arriver en retard dans ses cours.

Nous remercions encore une fois M. Giroudeau pour les points bonus qu'il va nous rajouter sur ce projet en échange de tous les journaux que nous lui apportions les jeudis matin.

Enfin, nous remercions nos parents, frères, soeurs, proches, et plus particulièrement le monsieur avec les lunettes bizarres de la pizzeria du RU.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Présentation</b>	<b>5</b>
2.1	Projet . . . . .	5
2.2	Plate-formes de développement . . . . .	5
2.2.1	Android . . . . .	5
2.2.2	iOS . . . . .	5
2.3	Organisation . . . . .	5
<b>3</b>	<b>Analyse</b>	<b>7</b>
3.1	Cahier des charges . . . . .	7
3.1.1	Menus . . . . .	7
3.1.2	Jeu . . . . .	7
3.1.3	Serveur . . . . .	8
3.2	Modélisation . . . . .	8
3.2.1	Général . . . . .	8
3.2.2	Menus . . . . .	8
3.2.3	Jeu . . . . .	12
3.2.4	Réseau . . . . .	12
3.3	Différences entre Android et iOS . . . . .	12
<b>4</b>	<b>Développement</b>	<b>13</b>
4.1	Mobile . . . . .	13
4.1.1	Menus . . . . .	13
4.1.2	Editeur de carte . . . . .	14
4.1.3	Jeu . . . . .	14
4.2	Serveur . . . . .	15
4.2.1	Json . . . . .	15
4.2.2	Servlet . . . . .	17
4.2.3	BDD . . . . .	20

<b>5</b>	<b>Manuel d'utilisation</b>	<b>21</b>
5.1	Menus . . . . .	21
5.2	Jeu . . . . .	21
5.3	Editeur . . . . .	21
<b>6</b>	<b>Discussion</b>	<b>22</b>
6.1	Problèmes . . . . .	22
6.1.1	Android . . . . .	22
6.1.2	iOS . . . . .	22
6.2	Améliorations . . . . .	22
6.2.1	Jeu . . . . .	22
6.2.2	Serveur . . . . .	22
<b>7</b>	<b>Conclusion</b>	<b>23</b>

# Chapitre 1

## Introduction

# Chapitre 2

## Présentation

- Nous + Tuteur

### 2.1 Projet

- On l'a proposé
- Principe du projet et description
- Rapprochement avec nos cours (I2A, Casar, Diwed)
- Buts / Attentes recherchés

### 2.2 Plate-formes de développement

#### 2.2.1 Android

- De qui ?
- Langage
- PC
- SDK - (le contenu)

#### 2.2.2 iOS

- De qui ?
- Langage
- Mac
- SDK - (le contenu)

### 2.3 Organisation

- SVN
- Réunions
- Répartition du travail

- Diagramme de Gant

# Chapitre 3

## Analyse

– Introduction

### 3.1 Cahier des charges

#### 3.1.1 Menus

Les menus se doivent d’être clairs et de rendre l’utilisation de l’application aisée. Il s’agit d’un jeu ne demandant aucune compétence particulière. Il va donc toucher un public large et doit pouvoir convenir à tout utilisateur. Cela passe d’abord par une navigation intuitive dans les menus.

Nous avons pour ce faire établi un diagramme d’activité reflétant les différents parcours possibles par un utilisateur lors de sa navigation dans l’application (voir section modélisation p9).

#### 3.1.2 Jeu

Le jeu est la partie la plus importante du projet. Il se décompose en trois parties : Le modèle, la vue et le contrôleur. Le modèle est composé du moteur physique, du moteur de rendu ainsi que de la hiérarchie de classe permettant de représenter l’ensemble des objets du jeu. La vue quant à elle est composée d’objets graphiques simples (Bouttons, images, ... ) et d’une partie représentant le jeu. Elle se doit d’être ergonomique et de permettre à l’utilisateur de pouvoir jouer très simplement. Le contrôleur permettra de faire le lien entre les actions de l’utilisateur sur le modèle. Cette décomposition permettra dans le futur de pouvoir modifier facilement le modèle et/ou la vue.

L’application se doit de pouvoir changer de langue, avec comme langues initiales le français et l’anglais. Elle doit permettre à l’utilisateur de jouer à des parties solitaires ou multijoueurs. Ce dernier possèdera un compte hors ligne et en ligne. Le premier permettra de personnaliser son profil comme par exemple pour modifier la couleur du joueur ou encore changer son pseudo... Il servira aussi à enregistrer les informations et les préférences de connexion sur une base de données locale, mais aussi les scores des joueurs (nombre de parties gagnées ou perdues). Le jeu devra permettre à l’utilisateur



de pouvoir créer différents comptes hors lignes en cas de partage de téléphone avec un ami ou un membre de famille, pour pouvoir garder en mémoire ses scores et ses préférences. Le compte en ligne quand à lui servira seulement à établir une connexion avec le serveur distant pour pouvoir jouer en multijoueur. Un menu d'aide doit apparaître pour pouvoir aider le joueur à comprendre le but du jeu et comment jouer. Ce dernier doit être simple et très explicite étant donné la large tranche d'âge d'utilisateur que vise cette application. Ensuite un éditeur de carte permettra aux utilisateurs de créer un large choix de cartes, grâce à une multitude de différents objets qui composeront les cartes. Ces dernières pourront être seulement utilisées en mode solitaire. Pour les parties solitaires une intelligence artificielle avec trois niveaux de difficulté devra permettre à un joueur débutant, intermédiaire ou confirmé de jouer comme bon lui semble pour pouvoir améliorer sa manière de jouer.

### 3.1.3 Serveur

Le serveur représente la partie réseau de notre projet. Il doit pouvoir rendre fonctionnel le jeu entre plusieurs téléphones (qu'ils soient de type iOS ou Android). Autrement dit il servira d'hébergeur pour les parties et il se chargera de faire s'interagir les joueurs, via leur mobile, entre eux. Nous parlons donc ici des parties multijoueurs.

Il devra être capable d'enregistrer des inscriptions de nouveaux joueurs, avec vérification qu'il n'y ait pas de doublons. Ces derniers seront inscrits dans la base de données du serveur. Les joueurs devraient ainsi pouvoir se connecter en utilisant le couple username / mot de passe, préalablement choisi. Suite à cela les utilisateurs seront à même de lister les parties en cours, ils pourront choisir de créer des parties ou de les rejoindre.

Voilà concernant les fonctionnalités qui ont été demandées pour la partie serveur.

## 3.2 Modélisation

### 3.2.1 Général

- Langage utilisé
- Tout en anglais
- Modèle Vue Contrôleur
- Documentation - (Javadoc / appledoc)

### 3.2.2 Menus

Au démarrage de l'application vous arrivez sur un menu d'accueil. Depuis celui-ci vous pourrez accéder à l'aide, à la liste des comptes locaux ou à la création d'un nouveau. Les menus de l'application ont été réalisés pour que l'utilisateur ait une utilisation intuitive de l'application. Ils se divisent en 4 grandes sections.

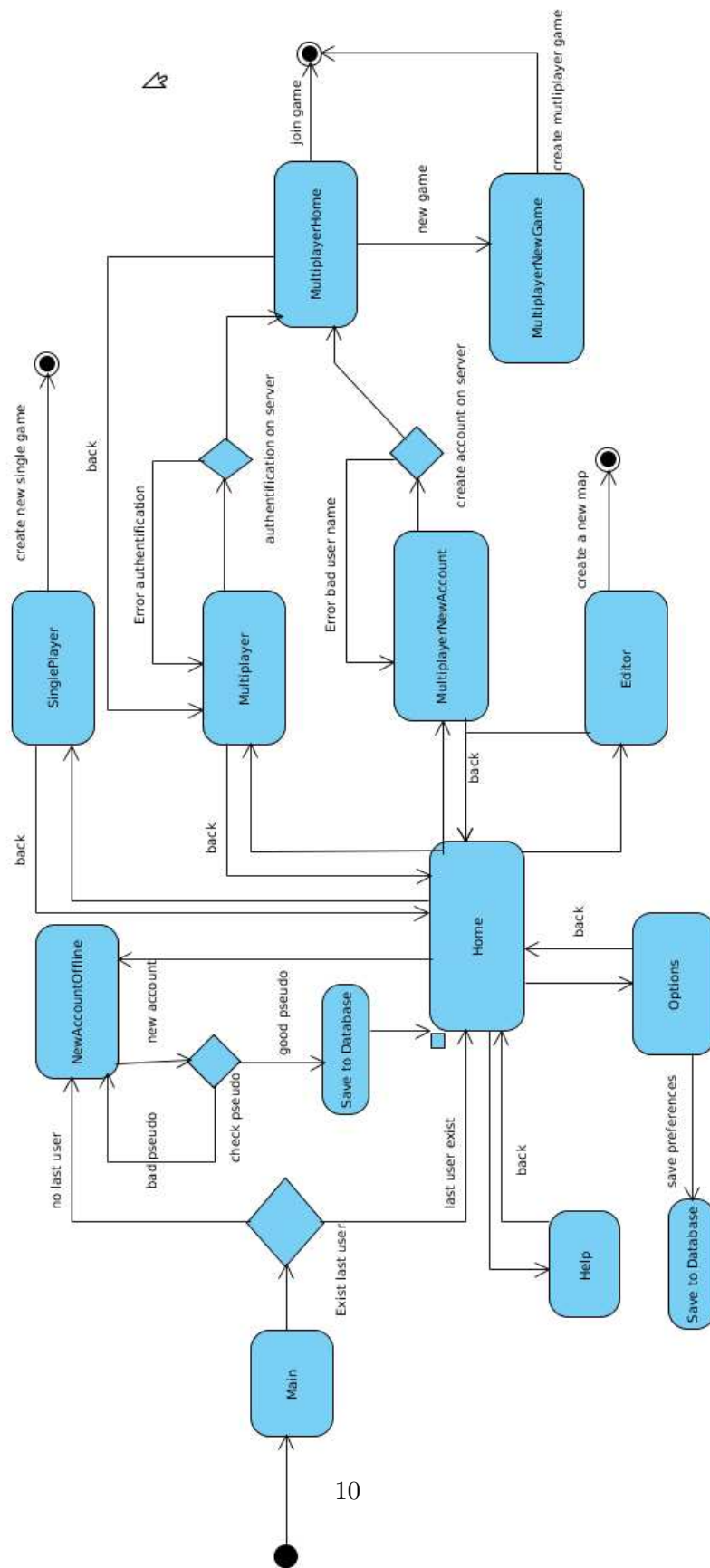
Vous avez tout d'abord la section de création de parties locales. Vous aurez accès à une liste de cartes ainsi qu'au réglage de difficulté des bots, leur nombre et le temps de

jeu. Le type de partie sera une fonctionnalité à venir. Vous n'aurez plus qu'à créer la partie configurée.

Dans la même catégorie se trouve la section des parties multijoueurs. En accédant à celle-ci vous allez pouvoir vous connecter à votre compte multijoueur, ou le créer si ne déjà fait. Vous accéderez ensuite à la liste des parties multijoueurs, que vous pourrez rejoindre, ou choisir de créer la votre. Dans le menu création le principe est proche des parties locales.

Suite à ces deux sections vient ensuite l'éditeur de cartes. C'est depuis ce menu que vous déciderez la création d'une nouvelle map de jeu local ou à l'édition d'une d'entre elles. Choisissez votre nom de carte et l'éditeur s'ouvrira ensuite à vous. Il vous sera possible à la fin d'enregistrer votre carte si vous désirez la conserver et l'utiliser comme carte de jeu.

Et enfin vient le menu des options. Depuis ce dernier vous pourrez gérer vos préférences systèmes telles que le volume ou la langue de l'application(anglais, français). Une sous-section de gestionnaire de profil est aussi présente. Une édition de vos comptes locaux, multijoueurs ou même vos paramètres de jeu comme la position du menu, sont modifiable depuis ce menu à onglets.



## **Base de données**

Une base de données locale a elle aussi été conçue. Cette dernière a pour but de stocker plusieurs types de données.

En effet dès lors qu'un compte local est créé sur le téléphone dans la table PlayerAccount, il est possible de conserver ses préférences de joueur tel que la couleur du joueur, le pseudonyme ou même ses paramètres de connexion multijoueur. Vous pourrez créer autant de comptes locaux que vous le désirez, et il sera possible d'éditer ou choisir son compte.

L'application est par ailleurs en mesure de conserver les valeurs sonores, la langue et même le dernier utilisateur de l'application, grâce à un son id qui est clé étrangère dans la table System(lastUser).

De plus l'application sera délivrée avec quelques cartes officielles, mais l'utilisateur aura libre droit de créer ses propres cartes de jeu via un éditeur. Elles seront alors stockées dans la table Map avec toujours une clé étrangère vers l'id de son créateur(owner).

## **Scénarios**

### 3.2.3 Jeu

- Diagramme classe
- Gameplay
- Gestion images / tile mapping
- Images
- Sons

### 3.2.4 Réseau

#### Serveur

Il a été fixé dans le cahier des charges que notre serveur devrait pouvoir effectuer plusieurs tâches particulières séparées. Nous avons donc décidé de les compartimenter en classes.

Les six éléments situés sur la partie haute du schéma ci-dessous(respectivement Inscription, Connection, GamesList, CreateGame, ConnectionGame et ManageGame), représente les différents tâches demandées. Elles sont reliées à une classe nommée contexte, qui leur permettra d'accéder aux mêmes données sans qu'il y ait de conflits. La partie basse représente les objets qui seront utilisés pour les parties en multijoueurs. Bien évidemment ces objets sont très proches de ceux utilisés dans les parties locales.

#### Base de données

Afin de pouvoir conserver les utilisateurs en lignes ainsi que leurs infos personnels pour permettre une authentification, nous avons dû établir une base de données sur le serveur. Cette dernière à été pensé comme demandé pour l'enregistrement de comptes. Une unique table nommée Users remplit donc cette fonction. Le serveur devra pouvoir y accéder en écriture(inscription) comme en lecture(connexion).

Elle ne comportera que deux champs, userName et password.

## 3.3 Différences entre Android et iOS

## Chapitre 4

# Développement

### 4.1 Mobile

#### 4.1.1 Menus

##### API et widget

TODO ludo

##### BDD

Après avoir effectué divers recherches, il s'est avéré que le gestionnaire de bases de données utilisé sur les deux plateformes est SQLite 3.

##### Android

Pour manipuler aisément les bases de données depuis l'application, nous avons créé une classe héritant de *SQLiteOpenHelper*. Cette dernière fournit des outils de manipulations. Un attribut *y* est instancié, il s'agit de la base de données elle-même, de type *SQLiteDatabase*.

Nous y avons créé 3 Tables, *PlayerAccount* sauvegardant toutes les informations sur les utilisateurs locaux, *System* concernant les propriétés du système, et enfin *Map* décrivant les informations relatives aux cartes de jeu créées par l'utilisateur.

Ainsi de nombreuses fonctions ont été implémentées dans le but de simplifier les interactions avec cette base de données depuis l'application. Il est par exemple possible de créer un nouvel utilisateur local, modifier ses préférences, gérer les configurations systèmes comme la langue ou le volume du son, ajouter de nouvelles maps ou même récupérer toutes les informations concernant un utilisateur.

Voici un exemple d'insertion d'un nouveau compte local dans la base de données. Rappelons que les tests d'existence du compte ont été faits depuis l'application même. Dans cet exemple vous verrez ainsi que nous commençons par récupérer les droits en écritures sur la base de données locale, puis nous créons un container qui servira à l'insertion

de valeur dans la base. Et enfin l'insertion est faite. Nous terminons tout de même en fermant l'accès à cette base.

Il s'agit là d'un schéma classique de fonction d'interaction avec notre base.

```
/** ajout compte hors ligne */  
public long newAccount(String nomCompte){  
    base = getWritableDatabase();  
  
    ContentValues entree = new ContentValues();  
  
    entree.put("pseudo", nomCompte);  
    long var = base.insert("PlayerAccount", null, entree);  
  
    base.close();  
    return var;  
}
```

## iOS

### Première utilisation

#### Création utilisateur

#### Gestion utilisateur

#### Gestion des préférences système

#### Création de carte (charger)

#### Création partie solo (tout)

#### Création partie multi (officielle)

### 4.1.2 Editeur de carte

#### Rendu

#### Interface utilisateur

#### Sauvegarde

### 4.1.3 Jeu

#### Moteurs

#### Rendu

#### Structure utilisée

- Pourquoi
- Avantages

## **Physique**

### **Structure utilisée**

### **Mouvements (collisions)**

### **Gestion des bombes**

– Threads

## **IA**

### **Pathfinding**

#### **A\***

#### **Aléatoire**

### **Prise de décision**

### **Interface utilisateur**

#### **Android**

#### **iOS**

## **4.2 Serveur**

Nous avons choisi de créer notre serveur sur une base de servlet. Ce fût ici aussi un point nouveau pour nous, réitérant les phases d'analyse, de découverte, de test et de mise en place. Le fonctionnement est basé sur les échanges de requêtes type HTTP, où à chaque demande correspond une réponse.

### **4.2.1 Json**

Soucieux des performances et de la rapidité des échanges entre applications et serveur, nous avons mis en place un protocole de communication client/serveur où les messages transitant sont des flux JSON. Ce dernier semblait être un format de données d'échanges optimal pour véhiculer le plus d'informations avec une taille moindre. De plus étant beaucoup utilisé, nos deux langages mettent à disposition des outils de sérialisation de leurs objets en JSON.



```
ServletInscription
Player => Serveur
{"username","password"}}
```

```
Serveur => Player
{"OK"} ou {"BU"}
```

```
ServletConnexion
Player => Serveur
{"username","password"}}
```

```
Serveur => Player
{"OK"} ou {"BU"}
```

```
ServletGameList
Player => Serveur
{"userKey"}
```

```
Serveur => Player
[{"class": "Game", "map": "mapName", "name": "gameName",
  "playerNumberConnected": nbConnected, "type": "gameType"}, {..}, {..}]}
```

```
ServletCreateGame:
Player => Server:
{"userKey": <userKey>, "game": {"name": <name>, "type": <type>, "map": <map>, "ennemiesNumber"
```

```
Server => Player:
{"OK"} ou {"errorType"}
```

```
ServletConnectionGame:
Player => Server:
{"userKey", "gameName"}}
```

```
Server => Player:
{[<1/2/3/4>, "play<true/false>", "map", "time<mm:ss>"]}
ou
{"errorType"}
```

```
ServletManageGame:
Player => Server:
{"userKey", "gameName", "action"}
```

```
Server => Players: (Player, bombs, blocs, score, time)
```

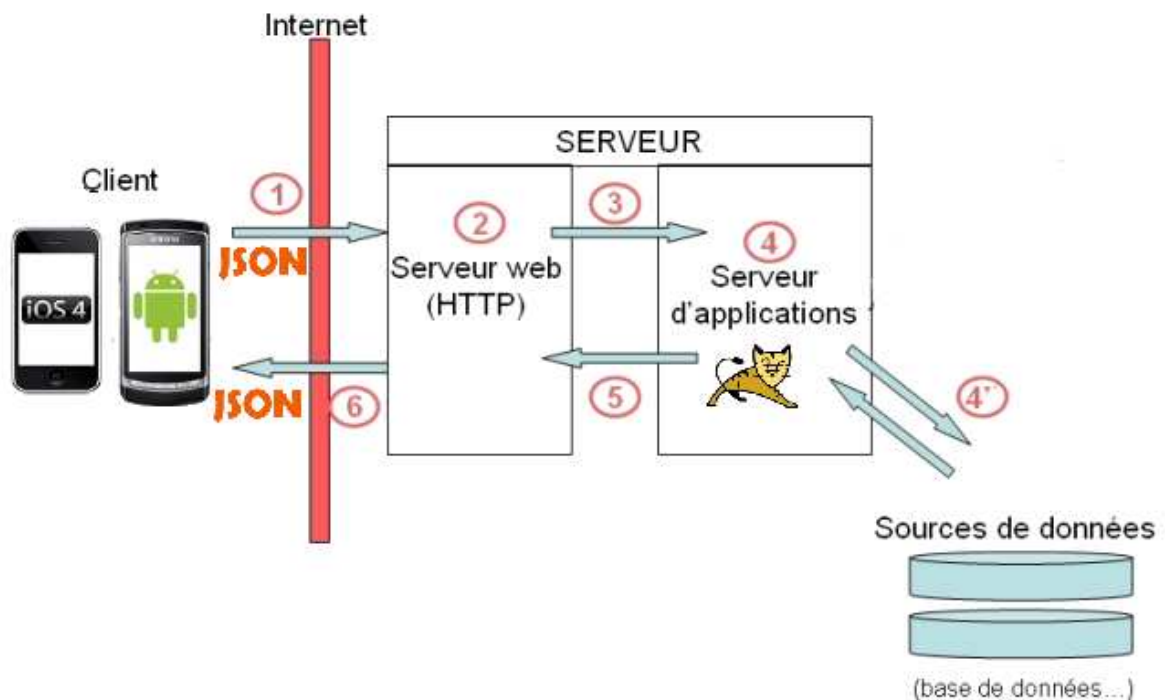
```
{[
  [ "x", "y", "direction", "dead <true/false>"],[...] ],
  [ "x", "y", "type", "explode <true/false>" ], [...] ],
  [ {"position": {"x", "y"}, "bonus": <bonus>}, [...] ],
  [1,2,3,4],
  "time <mm:ss>"]}]
ou
{"errorType"}
```

#### 4.2.2 Servlet

Comme il a été dit précédemment, notre serveur est accessible via des requêtes HTTP contactant des servlets. Ces servlets sont stockées dans un serveur d'application nommé Apache Tomcat. Il s'agit d'un conteneur libre de servlets Java 2 Enterprise Edition, mais il fait aussi office de serveur Web.

Une servlet est une classe Java qui permet de créer dynamiquement des données au sein d'un serveur HTTP. Une servlet s'exécute dynamiquement sur le serveur web et permet l'extension des fonctions de ce dernier, typiquement : accès à des bases de données.

#### Schéma de fonctionnement



1. Le client émet une requête pour demander une ressource au serveur. Par exemple la création de son compte multijoueur, qui pourrait se situer <http://Bomberklob.com/inscription>
2. Côté serveur, c'est le serveur web qui traite les requêtes HTTP entrantes. Il traite donc toutes les requêtes, qu'elles demandent une ressource statique ou dynamique. Seulement, un serveur HTTP ne sait répondre qu'aux requêtes visant des ressources statiques.
3. Ainsi, si le serveur HTTP s'aperçoit que la requête reçue est destinée au serveur d'applications, il la lui transmet. Les deux serveurs sont reliés par un canal, nommé connecteur.
4. Le serveur d'applications (dans notre cas Tomcat) reçoit la requête à son tour. Lui est en mesure de la traiter. Il exécute donc la servlet correspondante à la requête, en fonction de l'URL, en récupérant les valeurs dans le flux JSON entrant. Cette opération est effectuée à partir de la configuration du serveur, grâce un fichier `web.xml` faisant le mapping entre URL et servlet associée.  
La servlet est donc invoquée, et le serveur lui fournit notamment deux objets Java exploitables : un représentant la requête, l'autre représentant la réponse. La servlet exécute sa fonction et génère la réponse à la demande, sous forme de flux JSON. Cela peut passer par la consultation de sources de données, comme des bases de données (4' sur le schéma).

## En pratique

Le requetes font appel à la fonction post des servlet. Le flux entrant étant de type JSON, il faut désérialiser le flux dans un objet correspondant. Exemple l'utilisateur envoie son userName et son mot de passe crypté dans un tableau, sérialisé en JSON, pour pouvoir récupérer les informations nous procédons comme suit :

```
BufferedReader req =
    new BufferedReader(new InputStreamReader(request.getInputStream()));
OutputStreamWriter writer =
    new OutputStreamWriter(response.getOutputStream());
String message = req.readLine();

if (message != null) {
    response.setContentType("text/html");

    // désérialisation des infos de l'utilisateur dans une arraylist
    JSONDeserializer<ArrayList<String>> jsonDeserializer =
    new JSONDeserializer<ArrayList<String>>();
    ArrayList<String> identifiern;
    identifiern = jsonDeserializer.deserialize(message);

    username = identifiern.get(0);
    password = identifiern.get(1);

    ...}
```

## La sécurité

Ce serveur de jeu étant hébergé sur internet et contenant des informations sensibles d'utilisateurs, tels que des mots de passes, il était crucial d'instaurer des règles de sécurité et de cryptage.

En effet lors des inscriptions ou connexion au serveur pour le mode multijoueur, les mots de passes sont tout d'abord cryptés côté client et ensuite encapsulé dans un flux JSON, pour être envoyé au serveur. Il stockera ainsi la chaîne de caractères extraite de l'objet désérialisé. De cette manière à aucun moment les données confidentielles ne transiteront en clair.

De plus un mécanisme de session est en place. Dans la confirmation de connexion ou d'inscription, une userKey est générée. Elle correspond en réalité à l'identifiant de session envoyé par le serveur. Une fois associée à l'username correspondant, le tout est ajouté dans le tableau d'utilisateurs connectés. Cet userKey est ensuite nécessaire pour contacter les servlets suivantes. Si cet identifiant n'est pas envoyé ou n'est pas présent dans le tableaux des utilisateurs connectés, il sera alors impossible d'accéder aux ressources du serveur.

### 4.2.3 BDD

La base de données du serveur n'est pas très complexe. En effet elle ne fait qu'accueillir les couples `userName/password` des utilisateurs dans la table `Users`. Pour son accès, chaque servlet peut récupérer un objet de type `Connection`, instancié à l'initialisation du serveur. Il permettra à son tour de récupérer un objet de type `Statement`. L'application va l'employer pour transmettre des instructions à la base. Exemple d'insertion :

```
Statement theStatement = connection.createStatement();
theStatement.execute(
    "INSERT into Users VALUES ('"+ username +"','"+password+"')");
```

## Chapitre 5

# Manuel d'utilisation

### 5.1 Menus

TODO ludo

### 5.2 Jeu

### 5.3 Editeur

# Chapitre 6

## Discussion

### 6.1 Problèmes

#### 6.1.1 Android

#### 6.1.2 iOS

### 6.2 Améliorations

#### 6.2.1 Jeu

#### 6.2.2 Serveur

- Pooling
- Session

Chapitre 7

Conclusion