

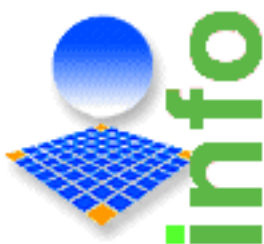
Ministère de l'Education nationale
Université de Montpellier II

Rapport de projet informatique GLIN601
Licence informatique 3^{ème} année
2011/2012



Rapport de projet tuteuré

Benjamin Maurin (chef de projet), Jessy Bonnotte, Mathieu Polizzi,
Steve Giner, Clement Agret, Renaud Legoc, Yohann Lam Seck, Paul Mura
Tuteur: Michel Meynard



Remerciements

Nous tenons à remercier en premier lieu notre tuteur de projet pour le soutien qu'il nous a apporté tout au long de notre projet. Il nous a également permis de découvrir les principes du travail en équipe ce qui nous permettra d'être polyvalents lors de notre entrée dans le monde de l'entreprise.

Nous tenons ensuite à remercier l'ensemble du corps enseignant de l'université de Montpellier 2 pour les connaissances qu'ils nous ont apportées. En effet, les différents cours que nous avons eus cette année nous ont permis de mener à bien ce projet.

Nous remercions tout particulièrement l'université Montpellier 2 qui nous a mis à dispositions des ordinateurs ainsi qu'une tablette afin de faciliter le développement. Sans cette aide, il nous aurait été très difficile d'optimiser notre travail.

| | |
|--|-----------|
| I. Présentation | 6 |
| II. Environnement..... | 7 |
| A) Présentation..... | 7 |
| 1) Présentation (jeux en ligne puis jeu de poker en ligne) | 7 |
| 2) Evolutions des jeux de poker en ligne | 7 |
| 3) Apport des jeux de poker en ligne | 8 |
| B) Etat de l'art..... | 9 |
| 1) Analyse du marché | 9 |
| 2) Inscription du projet dans le contexte de l'existant..... | 10 |
| III. Les outils utilisés | 11 |
| A) Le client C++ | 11 |
| 1) Présentation de C++ | 11 |
| 2) Le choix de Qt..... | 12 |
| 3) Présentation de Qt | 13 |
| 4) L'environnement de développement (Qt Creator) | 14 |
| B) Le client Android | 15 |
| 1) Le choix d'Android | 15 |
| 2) L'histoire d'Android | 15 |
| 3) Le principe de l'application Android..... | 16 |
| 4) Contexte d'utilisation | 16 |
| 5) Le SDK Android | 17 |
| 6) L'environnement de développement..... | 17 |
| 7) Présentation du XML..... | 18 |
| C) Le client JavaScript..... | 18 |
| 1) Le choix de javascript | 18 |
| 2) Présentation de JavaScript | 19 |
| 3) Principes du JavaScript | 20 |
| 4) L'environnement de développement..... | 20 |
| D) Le serveur | 21 |
| 1) Le choix du langage Java | 21 |

| | |
|---|---------------|
| 2) Présentation de Java | 21 |
| 3) Présentation de NoSQL | 22 |
| 4) L'environnement de développement Eclipse..... | 23 |
| E) Autres..... | 24 |
| 1) BOUML | 25 |
| 2) Doxygen..... | 26 |
| IV. Analyse | 27 |
| A) Problèmes listé dans le cahier des charges..... | 27 |
| 1) Fonctionnalités obligatoires | 27 |
| 2) Fonctionnalités optionnelles | 28 |
| B) Aspect fonctionnel de la partie Android | 28 |
| 1) Aspect fonctionnel de l'application Android..... | 28 |
| 2) Programmation de l'application Android..... | 36 |
| 3) Communication avec le serveur | 46 |
| C) Aspect fonctionnel de la partie C++ | 46 |
| 1) Présentation de l'application C++ | 46 |
| 2) Programmation de l'application C++..... | 47 |
| 3) Le dialogue avec le serveur | 48 |
| D) Aspect fonctionnel du client Web | 49 |
| 1) Aspect fonctionnel de l'application Web | 49 |
| 2) Programmation de l'application Web | 50 |
| E) Aspect fonctionnel de la partie serveur | 56 |
| 1) Communication et gestion des clients | 56 |
| 2) La gestion des jeux | 58 |
| 3) La base de données | 61 |
| 4) Le lecteur de commandes | 63 |
| 5) Diagrammes de l'application..... | 64 |
| F) Fonctionnement global du projet | 66 |
| 1) Communication Client-Serveur | 66 |
| 2) Déroulement d'une partie..... | 68 |

| | |
|---|---------------|
| V. Structure organisationnelle..... | 74 |
| A) Planning..... | 74 |
| B) Organisation et fonctionnement du groupe de travail | 75 |
| 1) Fonctionnement avec le tuteur..... | 75 |
| 2) Fonctionnement au sein du groupe | 75 |
| VI. Conclusion..... | 78 |
| A) Bilan..... | 78 |
| B) L'apport de ce projet | 78 |
| C) Améliorations possibles du logiciel..... | 78 |

I. Présentation

Le projet « Jeu en ligne » est réalisé par un groupe de huit étudiants de l'université des sciences de Montpellier : Jessy Bonnotte, Mathieu Polizzi, Benjamin Maurin, Steve Giner, Clément Agret, Renaud Legoc, Yohann Lam Seck et Paul Mura. Il nous a été assigné par M. Meynard dans le cadre de notre troisième année de licence à l'université de Montpellier 2. Il se déroulera durant le semestre 2: de Janvier 2012 à Mai 2012. Etant axé sur les jeux de poker en ligne, ce jeu est surtout destiné au particulier.

La finalité du développement sera de créer un serveur sur lequel différents types de clients pourront se connecter afin de faire des parties de poker. Ces clients seront de trois types :

- Client PC sous C++/Qt
- Client sous Java pour Android
- Client Web en JavaScript

Nous avons choisi ce sujet car il nous semblait intéressant par sa polyvalence : Il touche en effet à beaucoup de domaines que nous avons étudié tel que la programmation objet ou le réseau. De plus il nous permet de découvrir de nouvelles branches de l'informatique telle que la programmation sous Smartphone Android.

II. Environnement

A) Présentation

1) Présentation (jeux en ligne puis jeu de poker en ligne)

Un jeu en ligne est un jeu jouable par le biais d'un réseau informatique que ce soit par Internet ou en réseau local. Depuis la généralisation du haut débit dans les foyers, les jeux en ligne n'ont cessé de se multiplier et d'évoluer au cours des dix dernières années. Les jeux en ligne peuvent désormais incorporer de simples jeux d'écriture aux jeux complets et détaillés dans lesquels plusieurs joueurs se retrouvent d'une manière simultanée. De nombreux jeux en ligne se sont répartis en communautés virtuelles, transformant ainsi les jeux solos en forme d'activité sociale. La popularité grandissante de Flash et Java conduisent à une révolution sur Internet lorsque les internautes s'aperçoivent qu'ils peuvent accéder à des vidéos, musiques en streaming et un tas de nouvelles activités. Beaucoup plus tard, les sites pouvaient désormais offrir la disponibilité aux joueurs d'accéder à des jeux en réseau gratuits ou non. Certains jeux multi-joueurs en ligne comme World of Warcraft, Final Fantasy XI et Lineage II sont mensuellement payants pour accéder à leur service, tandis que d'autres jeux comme Guild Wars offre une accessibilité gratuite. Certains sites sont rémunérés en exposant leurs sponsors en ligne, tandis que d'autres, comme RuneScape, préfèrent laisser les joueurs jouer gratuitement et exposer de nouveaux contenus, payants pour les membres du réseau, à débloquent.

L'émergence du poker en ligne est liée à l'augmentation spectaculaire du nombre de joueurs à travers le monde. En 2005, les revenus des sites de poker en ligne ont été estimés à près de 200 millions de dollars par mois. En France, les activités du poker sont contrôlées par l'Autorité de régulation des jeux en ligne (ARJEL) qui délivre depuis le mois de juin 2010 des agréments aux sites de poker en ligne du domaine « .fr ».

2) Evolutions des jeux de poker en ligne

Le visage contemporain du poker en ligne se dessine en janvier 1998 avec la création de *Planet Poker*, le premier site de poker en ligne au sens moderne. Le véritable décollage du poker en ligne intervient quelques années plus tard, et plus précisément en 2001 avec l'avènement de *Paradise Poker*. Malgré le succès, les créateurs du site souhaitent prendre une autre voie et entreprennent de vendre la société pour un petit pactole de 15 millions de dollars. Mais c'est trop tard car la concurrence est déjà au rendez-vous puisqu'un nouveau venu, PokerStars, comprend vite que le succès du poker en ligne passera par une offre que les casinos ne peuvent pas proposer : des tournois de poker de plusieurs milliers de joueurs.

Le « main event » des « World Series Of Poker » est aujourd'hui un rendez-vous incontournable pour tous les passionnés de poker, joueurs comme observateurs, professionnels comme amateurs. Pourtant, le tournoi n'a pas toujours eu son rayonnement actuel. En 2003, ils étaient un peu plus de 800 sur la ligne de départ. Mais l'année suivante,

ils étaient près de 2 500, puis plus de 5 000 en 2005, jusqu'au record de 8 700 et quelques participants en 2006. Un évènement explique plus qu'aucun autre cette augmentation exponentielle : la victoire de Chris Moneymaker en 2003. Cet Américain au patronyme prédestiné, comptable de profession, s'était qualifié pour le tournoi sur Internet après n'avoir déboursé qu'une poignée de dollars.

En septembre 2006 est en effet promulgué l'Unlawful Internet Gambling Enforcement Act (UIGEA) : le Congrès américain décide tout simplement d'interdire les transactions bancaires à destination des sites de jeux en ligne. PartyPoker, à l'époque leader mondial, en tire les conséquences en quittant le marché états-unien. À l'inverse, d'autres sites comme PokerStars ou Full Tilt Poker choisissent de continuer à accepter les joueurs américains, se contentant de reporter leurs efforts marketing sur la promotion des versions gratuites de leur logiciel.



Après des années de flou réglementaire, le législateur français a décidé en 2010 l'ouverture du marché hexagonal des jeux en ligne. La loi du 12 mai 2010 a ainsi mis en place un cloisonnement des joueurs français sur les seules salles de poker en ligne disposant d'un agrément délivré par l'Autorité de Régulation des Jeux en Ligne (ARJEL).

Il existe désormais trois types de salles de poker en ligne :

- les rooms officielles qui ont demandé leur licence française et ouvert un site en ".fr". Leur site en ".com" est inaccessible aux joueurs français.
- les rooms qui n'ont pas demandé leur licence française, mais qui filtrent l'accès à leur logiciel afin d'être en conformité avec la loi.
- les rooms illégales, qui continuent d'accueillir les joueurs français sans demander de licence.

Les salles de poker en ligne sollicitant la délivrance d'un agrément auprès de l'ARJEL sont contrôlées strictement et doivent s'engager à respecter un cahier des charges précis. L'État ponctionne par ailleurs 2 % des mises sur chaque pot disputé, un prélèvement qui s'ajoute à celui réalisé par la room.

3) Apport des jeux de poker en ligne

L'avantage le plus évident du Poker en ligne est la capacité de jouer dans le confort de sa propre maison, avec l'atmosphère et l'environnement que le joueur désire. Le poker en ligne peut également être joué à toutes les heures de la journée, de la semaine et de l'année, et il est facile de jouer au poker online comme un passe-temps ou d'en faire une carrière, même lorsque l'on possède déjà un travail. Ainsi, jouer au poker en ligne impose nettement moins de dépenses que de jouer au poker dans les salles hors ligne. Il y a beaucoup de dépenses à jouer dans un casino terrestre, comme les frais d'inscription et de dépôt, sans oublier le temps et l'argent qui doivent être dépensés pendant les trajets. Jouer au poker en ligne peut être fait à votre convenance, sans avoir à organiser ou à s'imposer d'autres responsabilités.

Les salles de poker en ligne et les casinos offrent aussi un petit avantage de la maison, et donc, de meilleures chances de gagner. Les jeux sont automatisés, ainsi, plus de mains peuvent être joués par heure, et donc plus d'argent peut être gagné, c'est globalement plus avantageux. De plus, il existe de nombreux satellites en ligne et des tournois avec un record de prize pools et l'entrée dans certains des plus prestigieux tournois de poker du monde pour un buy-in minimum. La gamme de jeux est également plus riche et diversifiée et leur interface peut être modifiée en fonction du style que vous avez demandé, vous pouvez donc adapter le jeu de poker en ligne à vos envies et besoins.

Dans l'ensemble, le poker en ligne est une expérience du jeu de poker confortable et avantageuse, et recommandée par de nombreux professionnels et amateurs.

B) Etat de l'art

1) Analyse du marché

Actuellement le marché des jeux de poker en ligne est en plein essor, les entraves juridiques empêchant leur divulgation (poker avec argent réel) étant désormais plus permissives, de nombreuses sociétés ont investi le marché permettant à tous les types de joueurs de trouver ce qu'ils recherchent.

Afin d'illustrer ce marché nous avons choisi deux applications que nous avons analysé :

- *Zynga poker* qui est une application poker, elle permet le multi-joueurs via connexion avec le login de *Facebook* (destiné uniquement au partie de poker avec argent fictif). Elle dispose d'une interface (une fois connecté) permettant de choisir le type de partie auquel l'on souhaite participer (tournoi ou partie libre).

Tournoi : 8 joueurs le dernier en lice remporte l'argent (fictif) parié en début de parti (même somme pour tous les joueurs, plusieurs types de tournoi avec plus ou moins de jetons en jeu).

Parti Libre: accessible à tout moment par les autres joueurs sous réserve qu'il y est une place disponible. Les *recaves* (rajouter des jetons sur la table pour continuer à jouer) sont illimités sous réserve de disponibilité de crédit dans la banque du joueur. Sur chaque type de partie libre sont définis plusieurs variables : buy-in minimum-maximum.

On peut à tout moment quitter la partie en conservant les jetons gagnés pendant le temps de jeu. Il y a la possibilité d'avoir accès à une liste de joueurs « amis » (rejoindre une partie où participe un « ami »). De plus, deux systèmes de classement sont disponibles : un avec les joueurs « amis » est mis en place et un avec tous les joueurs de *Zynga*.

Un chat est mis à disposition qui, en plus de permettre aux joueurs de communiquer entre eux, rend compte des derniers événements de la partie. Chaque joueur dispose de 15 secondes pour effectuer l'action voulue à son tour (parier, checker, se coucher...).

Il est possible d'acheter des jetons quand le joueur est à court de jeton ou alors d'attendre 24h des jetons supplémentaires.

C'est un logiciel gratuit accessible sur *Facebook* via tous les navigateurs récents ainsi qu'*Android*, les appareils *Apple* et *Google+*.

- *Everest poker* est un logiciel pour jouer au poker en ligne, l'un des premiers sites de poker gratuit et traduit en français, il est possible de jouer avec de l'argent fictif ou réel, il y a possibilité comme sur *Zynga poker* de jouer en cash game et en tournoi ainsi qu'une possibilité supplémentaire de jouer en Sit & Go (partie libre avec partenaire fixe). Or ces détails, ce qui la différencie avec *Zynga poker* est son système de partie avec argent réel mais aussi le fait qu'elle ne soit accessible que par le téléchargement d'une application.

2) Inscription du projet dans le contexte de l'existant

Une importante phase dans notre projet a été tout d'abord d'analyser l'existant afin que notre application dispose de toutes les parties nécessaires au bon déroulement d'une « partie type » de poker pour ensuite ajouter des parties propres à notre vision du poker (modularité de l'application).

Notre application est cross-Platform, c'est-à-dire qu'elle permet à plusieurs types de clients de jouer entre eux (Android, JavaScript et C++).

De plus nous avons implémenté un système de création et de recherche de parties nous permettant de pouvoir rechercher une partie qu'un ami aurait créé ou alors de créer notre partie.

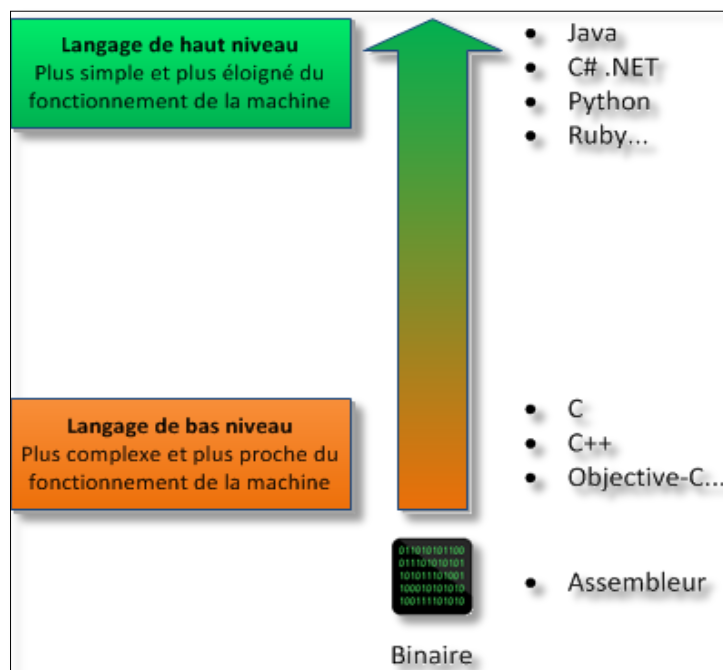
III. Les outils utilisés

A) Le client C++

1) Présentation de C++

En langage C, ++ est l'opérateur d'incrément. C'est pourquoi C++ porte ce nom. Le père de C++, **Bjarne Stroustrup**, a commencé à développer « *C with classes* » tout simplement le langage C avec des classes. En 1983, C++ est né.

Il existe une multitude de langages de programmation, que l'on compare selon de nombreux critères tels que le niveau du langage :



Graphique des différents langages

Comme on peut le constater C++ est considéré actuellement comme un langage de programmation dit de bas niveau. De plus C++ est un langage très puissant et particulièrement rapide, en effet il bénéficie d'une très grande popularité, particulièrement dans le monde des jeux parce qu'il allie puissance et rapidité.

Ainsi :

- le langage C++ est un des langages les plus populaires de la planète. Ce qui a l'immense avantage de procurer un très grand nombre de documentations, d'aide sur les forums, et de tutoriels très bien faits et fournis. De plus, il est incontournableement enseigné à la faculté.

- Il est très rapide et très puissant comme nous l'avons dit.
- Le langage C++ de Qt a également l'avantage d'être portable : le code source peut être changé en exécutable aussi bien sur Linux, Windows que Mac OS.
- Aussi le langage C++ possède de nombreuses bibliothèques.
- Enfin, C++ est un langage dit multi-paradigme : on peut aussi bien faire de la programmation procédurale, de la programmation générique que de la programmation orientée objet.

Le langage C++, bien que relativement ancien, continue à être amélioré. Une nouvelle version, appelée C++1x, est d'ailleurs en cours de préparation. De nombreux langages de programmation se sont par la suite inspirés du C++. C'est notamment le cas du langage Java. Comme quoi, un langage peut être vieux et rester d'actualité.

2) Le choix de Qt

Chaque système d'exploitation (Windows, Mac OS X, Linux...) propose au moins un moyen de créer des fenêtres. Le problème, c'est justement que ce moyen n'est en général pas portable.

Heureusement, il existe des bibliothèques multiplate-forme, cette méthode est plus souple et du coup la meilleure en générale. Cette méthode en plus de son avantage de portabilité, permet du coup d'assurer la durabilité du programme généré et assure ainsi un certain confort.

Parmi les bibliothèques multiplate-forme, on trouve entre autres GTK+ et Qt :

- GTK+ est une des plus importantes bibliothèques utilisées sous Linux. Elle est portable et est utilisable en C. Néanmoins, il existe une version C++ appelée GTKmm. C'est la bibliothèque utilisée par Firefox par exemple.
- Qt permet la portabilité des applications qui n'utilisent que ses composants par simple recompilation du code source. Les environnements supportés sont les Unix (dont Linux) qui utilisent le système graphique X Window System, Windows et Mac OS X. Le fait d'être une bibliothèque logicielle multiplate-forme attire un grand nombre de personnes qui ont donc l'occasion de diffuser leurs programmes sur les principaux OS existants. Qt est notamment connu pour être la bibliothèque sur laquelle repose l'environnement graphique KDE

L'importante popularité de Qt à l'avantage d'assurer un nombre plus important de personnes le connaissant. De plus, les tutoriels sont plus nombreux et plus fournis sur internet et particulièrement au sein de la communauté francophone. Enfin l'environnement de développement QtCreator, jouissant également d'une grande popularité, est plus simple d'utilisation et également plus fourni en tutoriels et forum d'entraide. C'est la qualité de sa bibliothèque, de sa documentation, et sa popularité donc qui ont défini le choix de Qt.

D'ailleurs, Qt est utilisée par de nombreuses entreprises. Du coup, Qt est à l'origine de nombreux GUI tel que ceux d'Adobe Photoshop Elements, de Google Earth ou encore de Skype ...

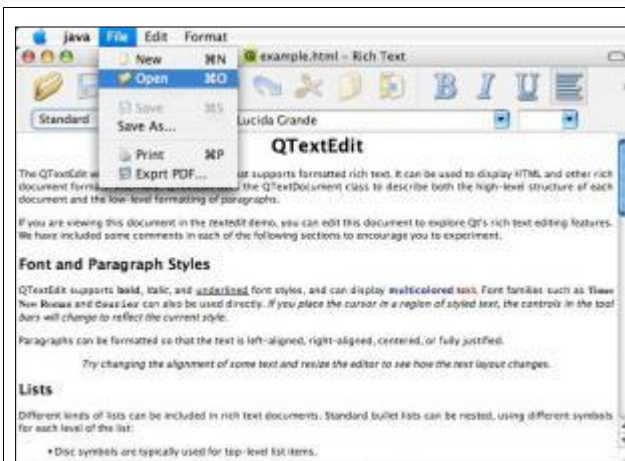
3) Présentation de Qt

Comme précisé précédemment, Qt est une bibliothèque multiplate-forme pour créer des GUI. Qt est écrite en C++ mais il est aujourd'hui possible de l'utiliser dans d'autres langages comme Java, Python, etc.

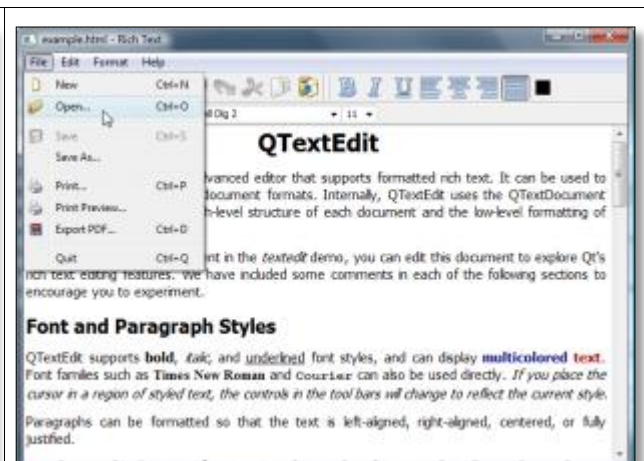
En fait, Qt est plus un ensemble de bibliothèques, si bien qu'on parle de **Framework** : un ensemble d'outils pour développer.

Qt (prononcé officiellement en anglais cute (/kju:t/) mais erronément couramment prononcé Q.T.) est un Framework orientée objet et développé en C++ par Qt Development Frameworks, filiale de Nokia. Il offre des composants d'interface graphique (widgets), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, etc.

La bibliothèque embarque divers thèmes de widgets qui lui donnent une bonne intégration visuelle sur toutes les plateformes. Sur les environnements de bureau GNOME, Mac OS X et Windows les applications Qt ont ainsi l'apparence d'applications natives :



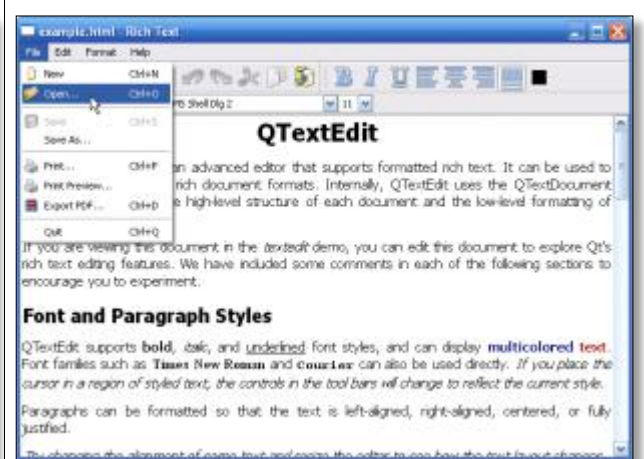
Sous Mac OS X



Sous Windows 7



Sous Linux



Sous Windows XP

Qt signifie "Cute" ce qui signifie "Mignon", parce que les développeurs trouvaient que la lettre Q était jolie dans leur éditeur de texte. À propos de la licence de Qt : Qt est distribué sous deux licences, LGPL ou propriétaire.

Le projet d'environnement graphique KDE a dès le début utilisé la bibliothèque Qt. Mais avec le succès de cet environnement, une certaine partie de la communauté du logiciel libre a critiqué la licence de Qt qui était propriétaire et incompatible avec la GNU GPL utilisée par KDE. Ce problème fut résolu par la société Trolltech qui mit les versions GNU/Linux et UNIX de Qt sous licence GNU GPL lorsque l'application développée était également sous GNU GPL. Pour le reste, c'est la licence commerciale qui entre en application.

Créé en juin 1998, la fondation *KDE Free Qt Foundation* est chargée de s'assurer de la disponibilité de Qt pour le développement de logiciels libres. Le rachat de Trolltech par Nokia le 28 janvier 2008 ne remet pas en cause la politique de double licence, l'entreprise finlandaise soutient même KDE.

Le 18 janvier 2008, Trolltech annonce que les versions 3 et 4 de Qt sont à partir de cette date sous licence GPLv2 et GPLv3. Un an plus tard, le 14 janvier 2009, Trolltech annonce qu'à partir de Qt 4.5, Qt sera également disponible sous licence LGPL v2.1. Cette nouvelle licence permet ainsi des développements de logiciels propriétaires, sans nécessiter l'achat d'une licence commerciale auprès de Qt Development Frameworks.

En Mars 2011, Nokia cède l'activité services et gestion des licences commerciales de Qt à la société Digia.

4) L'environnement de développement (Qt Creator)

Qt Creator est un environnement de développement intégré multiplate-forme faisant partie du Framework Qt. Il est donc orienté pour la programmation en C++.

Il est particulièrement optimisé pour développer avec Qt. En effet, c'est un programme tout-en-un qui comprend entre autres :

- Un IDE pour développer en C++, optimisé pour compiler des projets utilisant Qt (pas de configuration fastidieuse)
- Un éditeur de fenêtres, qui permet de dessiner facilement le contenu de ses interfaces à la souris
- Une documentation très complète sur Qt

Il intègre directement

- dans l'interface un débogueur
- ainsi qu'un outil de création d'interfaces graphiques
- L'éditeur de texte intégré permet l'autocomplétion ainsi que la coloration syntaxique.
- Qt Creator utilise MinGW par défaut sous Windows.

B) Le client Android

1) Le choix d'Android

Il existe aujourd'hui sur le marché des systèmes d'exploitation pour Smartphones cinq grands acteurs :

- Nokia avec son système d'exploitation Symbian
- RIM avec BlackBerry OS
- Microsoft avec Windows Phone 7
- Apple avec IOS4
- Google avec Android

Ces dernières années, Android a eu une croissance considérable (+886 % en un an) et cela devrait continuer, les différentes études montrent qu'Android devrait être leader des systèmes d'exploitation mobiles en 2012. Cela nous permet de croire à un avenir pour ce projet. De plus, Android est basé sur un noyau linux, ainsi, Android est libre et ouvert. Cela permet un développement aisé et peu onéreux.

Enfin, Android utilise le couple de langage XML – Java pour le développement. Cela nous permet de mettre en œuvre des langages utilisés au cours de notre DUT tout en utilisant un nouveau SDK. Nous avons décidé d'utiliser la version 1.6 d'Android (appelée Donut) pour notre développement. Ce choix a été fait en raison du matériel disponible et par la volonté de diffuser notre application à un maximum de personnes possible. En effet, Android 1.6 est aujourd'hui encore très utilisé et les outils de développement associés permettent une conception aisée.

2) L'histoire d'Android

En juin 2005, Google rachète une start-up du nom d'Android, spécialisée dans le développement d'application mobile. C'est là que commence le développement de leur propre système d'exploitation mobile. Google présente au public ce système pour la première fois en 2006 mais il faudra attendre 2007 pour avoir l'annonce officielle.

C'est seulement fin 2008 que le premier téléphone utilisant le système d'exploitation Android arrive sur le marché américain : c'est le G1 sorti le 22 octobre. Ce téléphone utilise la version 1.1 d'Android. C'est aussi en 2008 que Google présente l'Android Market : un magasin d'application pour téléphone sous Android tel l'AppStore pour l'iPhone (sorti en 2007). Ce magasin permet de télécharger directement de petites applications afin de décupler les possibilités du téléphone en utilisant toutes les ressources des appareils (GPS, accéléromètre etc....)

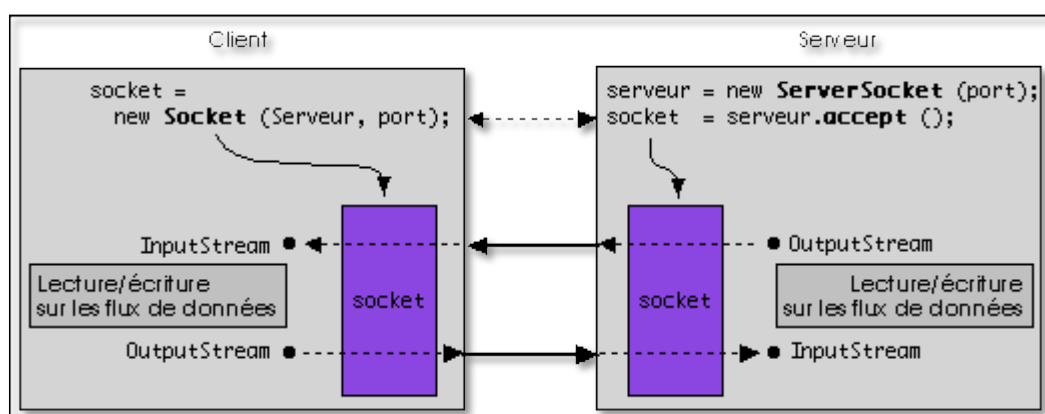
Google continuera à publier assez fréquemment de nouvelles versions d'Android (quatre en deux ans et qui portent toutes le nom d'un dessert et suivent l'alphabet) et les terminaux mobiles se multiplient. En 2009, Google a décidé de distribuer son propre téléphone, le Nexus One afin de stimuler le développement d'Android dans le monde. Il récidivera en décembre 2010 avec le Nexus S puis en 2011 avec le Galaxy Nexus.

Aujourd'hui, Android est en version 2.3 et Google prévoit de sortir la version 3.0 pour le début de l'année 2011. On compte 160 000 téléphones sous Android activés chaque jour dans le monde et la barre des 100 000 applications a été atteinte dans l'Android Market fin juillet 2010.

3) Le principe de l'application Android

Pour notre projet, nous devons permettre à notre téléphone de communiquer avec le serveur gérant les XBee. Pour ce faire nous avons choisi d'utiliser une connexion par socket. Un socket est une sorte de tube, si celui-ci relie un serveur à un client, cela permettrait de communiquer en mettant au point un protocole assez simple. L'utilisation des sockets étant similaire à celle de flux, nous n'aurons qu'à mettre nos instructions dans ces flux pour que le serveur les reçoive. Nous aurons donc besoin de deux processus : un pour « l'écriture » qui permettra à notre application d'envoyer des informations au serveur et un pour la « lecture » qui lui, tournera en permanence.

Sur l'exemple suivant, on peut voir comment se passe la connexion entre le client et le serveur. Nous créons d'abord notre serveur qui est relié à un port du réseau puis on lui autorise les connexions par socket. Notre application Android crée ensuite un socket relié au serveur. Nous créons ensuite nos flux d'entrée et de sortie afin de pouvoir communiquer avec le serveur. Nous appellerons une méthode « *write(String message)* » qui permettra d'écrire dans le flux de sortie le message à destination du serveur. Pour l'écoute, nous créerons un processus qui lui, écoutera en permanence sur le flux d'entrée tout message en provenance du serveur.



4) Contexte d'utilisation

Comme nous l'avons vu précédemment, notre projet s'inscrit principalement dans les jeux en ligne. Notre application sera tout à fait conforme à cet environnement. En effet, l'intérêt de développer notre application sur Smartphone permettra une utilisation complètement nomade : l'utilisateur pourra jouer sa maison quel que soit l'endroit où il se trouve (que ce soit depuis son canapé ou depuis son lieu de travail). De plus, l'idée d'un appareil mobile tout-en-un se concrétise un peu plus avec cette application.

5) Le SDK Android

Afin de faciliter le développement sous Android, Google met à disposition gratuitement un SDK. Cette boîte à outils, basée sur le langage Java propose tout le nécessaire pour développer des applications sous Android :

- Les bibliothèques nécessaires au développement.
- Une documentation complète.
- Des drivers pour les téléphones afin d'installer rapidement les applications et de les tester sans passer par l'Android Market.
- Un système de gestion d'émulateurs de téléphones sous Android.
- Des outils de débogages.



L'émulateur Android fourni avec la JDK

Le SDK, bien que basé sur le langage Java comme dit précédemment, utilise le langage XML pour construire les interfaces des applications.

6) L'environnement de développement

NetBeans (appelé à l'origine Xelfi), a vu le jour en tant que projet d'étudiant en République Tchèque, en 1996. Le but était d'écrire un EDI Java semblable à Delphi, mais en Java. Une compagnie fut créée autour de ce projet, nommé NetBeans. Il y a eu deux versions commerciales de NetBeans, appelées Developer 2.0 et Developer 2.1. Aux alentours de mai 1999, NetBeans sortit une version bêta de ce qui aurait dû être Developer 3.0. Quelques mois plus tard, en octobre 1999, NetBeans fut racheté par Sun Microsystems. Après quelques temps de développement supplémentaire, Sun sortit l'EDI Forté Fro Java, Edition Communauté - le même EDI qui avait été présenté en bêta comme NetBeans Developer 3.0.

Il y a toujours eu un intérêt pour l'open-source chez NetBeans, c'est pourquoi, en juin 2000, Sun mis l'EDI NetBeans en open-source.

NetBeans présente aujourd'hui de nombreux atouts justifiant son utilisation au sein de ce projet :

- Il est simple d'utilisation.
- Nous sommes familiarisés avec son interface car nous l'avons déjà utilisé.
- La documentation est riche notamment car NetBeans est développé par Sun, et c'est Sun qui a mis au point Java.
- Le produit est de qualité et a fait ses preuves.
- Les mises à jour sont fréquentes et le logiciel ne fait que s'améliorer.

7) Présentation du XML

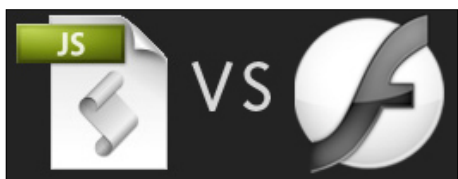
Lors de la conception de notre application Android, nous aurons besoin d'utiliser le langage XML. Il sera utilisé dans le « manifeste » de l'application et dans le « main.xml ».

XML (Extensible Markup Language, langage extensible de balisage) est un langage permettant la définition d'objets. Il a vu le jour pour sa version 1.0 en 1998 et la version 1.1 a été publiée en 2004. Un document XML peut être vu comme un arbre : le fichier contient un ou plusieurs objets principaux qui eux-mêmes contiennent d'autres objets et ainsi de suite. L'intérêt du XML est que l'utilisateur peut définir lui-même tous ces objets. Nous verrons dans notre application comment se servir du XML pour créer nos objets. La syntaxe est très précise, un saut de ligne avant la déclaration du document retournera une erreur lors du passage du code au parseur.

Voici une anecdote concernant le XML : le groupe de travail XML ne s'est jamais rencontré face-à-face, la conception a été réalisée en utilisant courriers électroniques et téléconférences hebdomadaires. Les principales décisions de conception ont été prises en une vingtaine de semaines de travail intense entre juillet et novembre 1996, lorsque le premier travail de spécification XML a été publié. D'autres travaux de conception sont poursuivis jusqu'en 1997, puis le XML 1.0 est devenu une recommandation W3C le 10 février 1998.

C) Le client JavaScript

1) Le choix de javascript



Parmi les trois clients imposés dans ce projet, l'un d'entre eux devait être accessible depuis un navigateur internet. Nous cherchions alors un langage capable de gérer aisément un page html, gérant des animations, mais aussi pouvant communiquer avec le serveur.

Il nous est venu alors en tête deux langages : JavaScript et FLASH.

Mais notre choix s'est porté sur le premier car d'abord, il est simple d'apprentissage, puis disponible en natif sur tous les navigateurs contrairement au FLASH qui nécessite une installation. Ensuite les applications développées avec ce langage sont moins lourdes que celles développées en FLASH (ces-dernières sont notamment gourmandes en ressources...), donc moins de temps de chargement.

Voici une liste résumant les raisons de notre choix :

- langage proposé par notre tuteur (il n'est jamais mal d'aller dans son sens)
- langage facile à apprendre
- langage puissant, nombreuses possibilités de programmation
- communauté de développeurs conséquente (on trouve de l'aide facilement)
- applications légères
- applications moins gourmandes en ressources que les applications FLASH
- langage non propriétaire
- disponible en natif sur tous les navigateurs internet
- gestion d'une page html aisée
- gestion des animations peu complexes très simple

2) Présentation de JavaScript



C'est un langage de programmation de scripts. C'est à dire dans ce cas-ci qu'il est directement interprété par le navigateur internet, sans passer par des phases de compilation et d'édition de liens comme les langages plus traditionnels comme C ou C++. On l'utilise le plus souvent pour manipuler le contenu d'une page html et ainsi la rendre dynamique. C'est un langage orienté objet à prototype. Autrement dit on y manipule des objets qui ne sont pas des instances de classes. Pourtant ces objets sont instanciés par des constructeurs. On peut y intégrer des attributs et des méthodes. A la différence des langages objets à classe, le code source d'un prototype est changeable Le prototype permet notamment de créer des objets héritiers personnalisés.

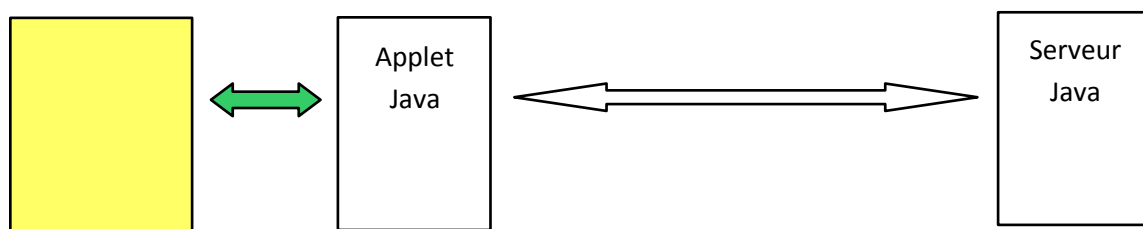
Il a été créé en 1995 sous l'inspiration de Brendan Eich qui travaille alors pour Netscape Communications Corporation. Il s'inspire de nombreux notamment Java. Mais simplifie la syntaxe pour rendre le langage plus séduisant aux yeux des débutants.

Initialement développé pour des scripts côté serveur il est un peu avant sa sortie retravaillée permettre le développement de script côté client. Il fut d'abord nommé LiveScript mais fut renommé JavaScript car Netscape Communications Corporation était alors en partenariat avec Sun Microsystems, créateur de Java. Ce qui permit au nouveau langage de bénéficier de la notoriété de Java alors en plein essor. Il est présenté comme un

complément à Java et créer ainsi la confusion entre les deux langages auprès des utilisateurs. Bien qu'ayant des similitudes syntaxiques, ces deux langages sont fondamentalement différents.

Il sort en décembre 1995. En mars 1996, NetScape intègre le moteur JavaScript à son navigateur internet au succès grandissant. Ainsi JavaScript est rapidement adopté dans le développement web côté client. Face à ce succès, Microsoft réagit en développant en août 1996 JScript qu'il intègre à son navigateur « Internet Explorer » estampillé 3.0.

3) Principes du JavaScript



Le but premier de l'application est d'afficher les différents événements de la partie. Il y a donc une première partie de l'application qui s'occupe de l'affichage.

En plus de l'affichage, le client doit pouvoir communiquer avec le serveur (écrit en Java). Devant la difficulté de dialoguer via une socket directement entre le client JavaScript et le serveur Java de manière directe, nous avons décidé d'utiliser un applet Java, à exécuter à chaque ouverture de la page, qui fera le lien entre nos deux applications. Cette applet s'appelle `JavaSocketBridge`. Elle établit un pont et ainsi transmet au client les instructions et autres informations que le serveur envoie. Inversement elle communique au serveur les informations envoyées par le client.

Enfin, il y a une phase d'identification/création de compte qui va permettre d'être reconnu par le serveur. Une partie de code permet donc la gestion des informations de l'utilisateur.

4) L'environnement de développement

Nous avons dans un premier temps décidé de travailler sur un unique système d'exploitation : Windows. Mais parmi les deux développeurs responsables du client JavaScript, l'un était sur Mac OS, l'autre sur Windows. Mais JavaScript est multiplateforme et rares sont les instructions non comprises par tous les navigateurs internet à la fois et a fortiori les 3 systèmes d'exploitations principaux. Ainsi pour rendre notre application potentiellement accessible par encore plus de joueurs, nous avons décidé de rester dans cette configuration.

Pendant l'élaboration du code, nous avons privilégié le navigateur Google Chrome, mais nous avons testé avec succès notre client sur Safari et Firefox.

L'édition du code s'est faite avec Notepad++ sur Windows et TextWrangler sur Mac OS. Cependant, un simple éditeur de texte aurait suffi, ces derniers n'apportant que la coloration des syntaxes réservées.

D) Le serveur

1) Le choix du langage Java

Nous avons choisi Java pour différentes raisons :

- C'est un langage qui permet de programmer avec moins d'erreur qu'un langage comme le C++.
- C'est un langage multiplateforme, ce qui permet de reprendre le code sans aucun problème.
- C'est un langage orienté objet qui permet d'implanter facilement des applications multitâches.
- En ce moment nous étudions le C++ en cours et le projet nous permet de ne pas perdre la main avec le Java et même mieux observer les différences entre ces deux langages.
- Et pour finir, étant donné que les trois autres clients dépendent de notre avancement, il était plus raisonnable d'utiliser un langage plus structuré, sur lequel nous sommes plus compétents et avec lequel nous avons déjà expérimenté le multitâches sérieusement.



Logo de Sun

2) Présentation de Java

Java est un langage de programmation né en 1995. Il a été par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982).

C'est un langage orienté objet semi-interprété qui est facilement portables sur différents systèmes d'exploitation tel qu'UNIX, Windows, Mac OS ... Sa portabilité est garantie par une plate-forme spécifique à chaque système.

Cette plate-forme est la [machine virtuelle](#) qui effectue la [traduction](#) et l'exécution du bytecode (code intermédiaire plus proche des instructions machines que le code source) en code natif.

3) Présentation de NoSQL

➤ Présentation rapide :

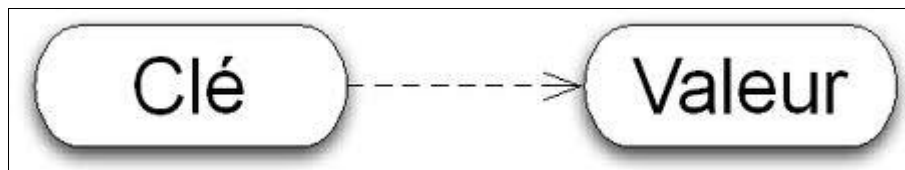
Le noSQL est une catégorie de SGBD non-relationnelle et utilisé par de nombreux acteurs de l'informatique tels que Google, Amazon.com, Facebook ou eBay.

Il existe plusieurs architectures de base de données noSQL, mais la plus courante et la plus simple est le système de clé-valeur. Les données sont simplement représentées par un couple clé/valeur. La valeur peut être une simple chaîne de caractères, un objet sérialisé... Les valeurs sont ensuite retrouvées comme dans une table de hachage grâce à sa clé.

Le SGBD choisit pour le projet est de type document, c'est-à-dire qu'il ajoute au modèle clé-valeur l'association d'une valeur à un ensemble d'autres valeurs.

La structure orientée document permet :

- Ajout, modification, lecture ou suppression de seulement certains champs dans un document.
- Indexation de champs de document permettant ainsi un accès rapide sans avoir recours uniquement à la clé.
- Requêtes élaborées pouvant inclure des prédicats sur les champs.



Exemple d'association

➤ Pourquoi le noSQL ?

- Simplicité : pas de tables à créer, pas de conception préliminaire et rajout de nouveaux types de valeurs sans problèmes.
- Performance : la complexité pour accéder à une valeur dans la base de donnée est presque constante.
- Montée en charge élevée : Même en rajoutant un grand nombre de valeurs, la base de données reste performante et fiable.

➤ Choix du SGBD : MongoDB :

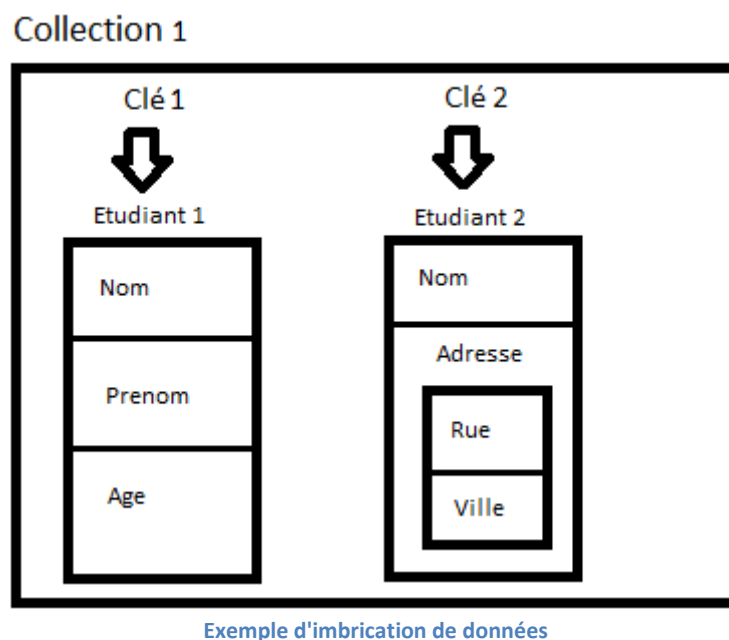


MongoDB est un SGBD noSQL libre développé depuis 2007 par 10gen. Il permet de manipuler des objets structurés au format BSON (JSON binaire).

Concrètement les données prennent la forme de documents enregistrés dans des collections, une collection pouvant ainsi contenir un nombre quelconque de documents. Si on souhaite faire une comparaison avec les bases de données SQL, on peut dire que les collections sont l'équivalent des tables, et que les documents sont l'équivalent des enregistrements.

Cependant il y a une différence fondamentale entre les deux approches : si les enregistrements d'une table SQL et les documents d'une collection MongoDB se composent tous deux de champs, dans les bases SQL chaque enregistrement de la table contient exactement les mêmes champs, dont seul le contenu varie.

A contrario dans une collection MongoDB, les documents peuvent avoir des champs totalement différents. De plus, il est possible d'imbriquer des documents. (Un champ peut contenir plusieurs champs).



➤ Pourquoi MongoDB ?

- Libre et gratuit : idéal pour un projet étudiant.
- Simplicité et documentation : MongoDB possède une large documentation et une API simple à utiliser, permettant de vite programmer et de pouvoir faire reprendre la gestion de la base de données par d'autres personnes.
- Performance et amélioration : la structure orienté document permet d'avoir de très bonnes performances même dans le cas d'une complexification ultérieure de la base de données.

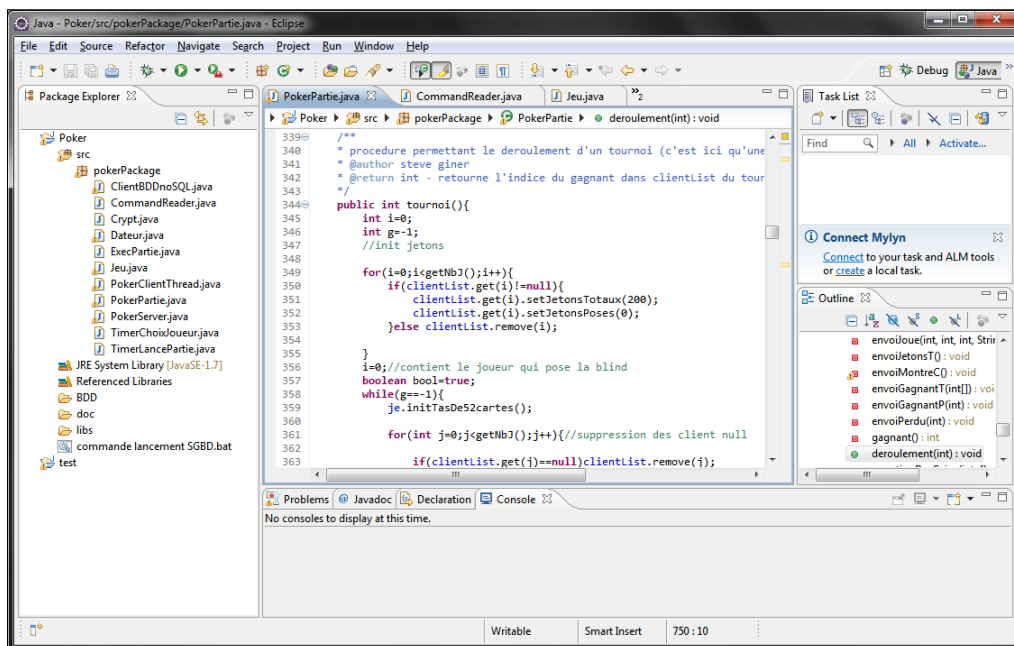
4) L'environnement de développement Eclipse



Eclipse est un environnement de développement (IDE) libre, extensible, universel et polyvalent. Ceci permet donc au programmeur de disposer de différents outils et d'une interface pour développer plus facilement et rapidement des applications.

Nous avons choisi Eclipse pour plusieurs raisons :

- Tout d'abord, c'est un outil que nous avons utilisé durant notre cursus, on est donc familiarisé avec son interface et ses fonctionnalités et sûr de sa fiabilité.
- Il est libre, gratuit et dispose d'un plugin Java.
- Il permet de générer une Javadoc d'une très bonne qualité en quelques clics.
- Il permet l'auto-complétion, la coloration et correction du code en direct, de compiler et d'exécuter plus simplement que dans un terminal et d'accéder plus rapidement aux erreurs dans le code à l'aide de lien dans la console. Il permet aussi de générer des codes génériques tel que le 'try/catch' pour la gestion des exceptions ou les importations de librairies.
- Il permet aussi la gestion de projet, paquetage et l'importation de librairies supplémentaires en quelques clics.



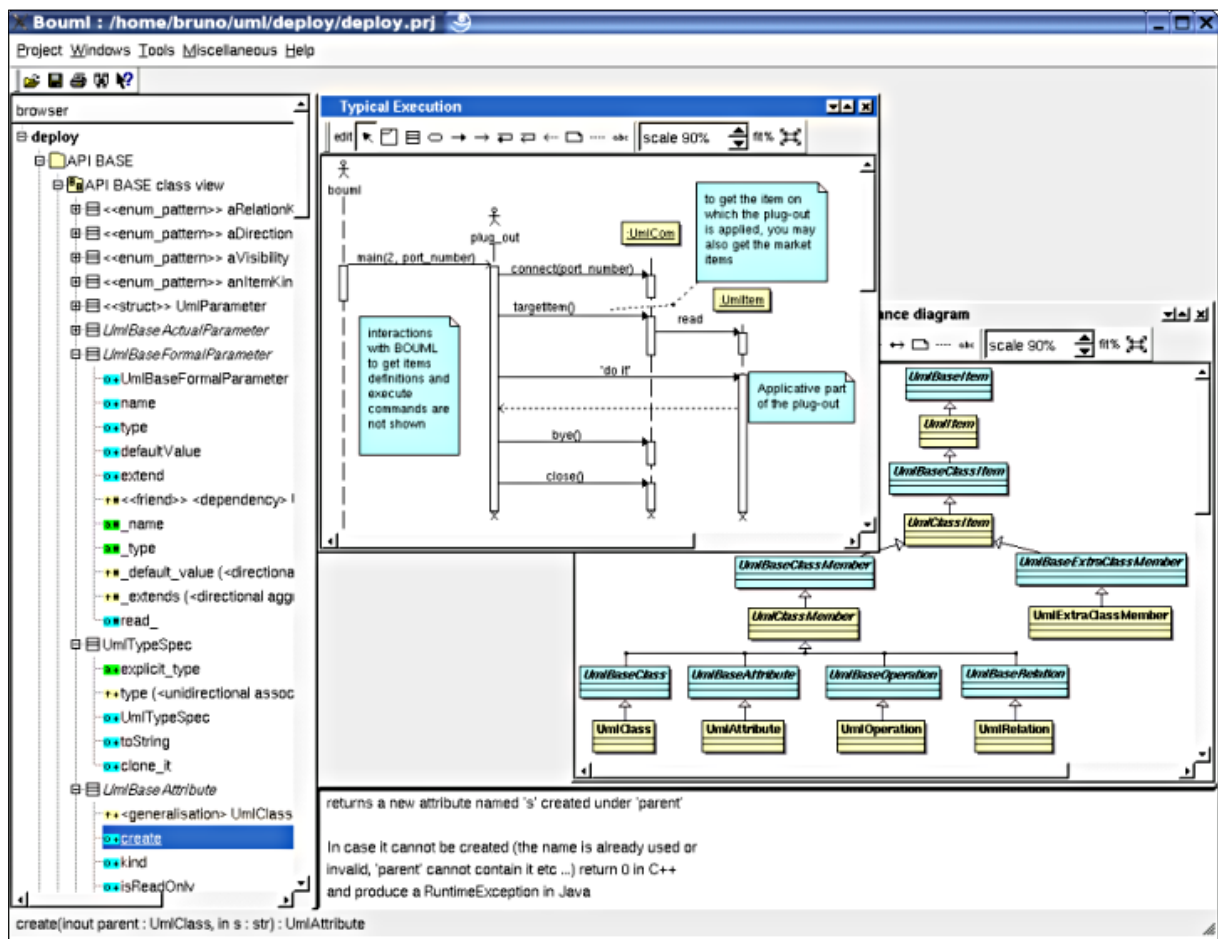
Fenêtre principal d'Eclipse

E) Autres

1) BOUML



BOUML est un logiciel de création de diagrammes UML. Programmé en C++ et Qt, il est distribué sous licence propriétaire depuis la version 5.0. Il est multilingue, supporte la génération de code et la rétro-ingénierie. BOUML est un logiciel gratuit de création de diagrammes UML programmé en C++ et Qt. Ce logiciel permet de dessiner des diagrammes suivant la norme UML 2.0 et est également capable d'effectuer de la rétro-ingénierie : c'est-à-dire qu'il permet de générer des diagrammes UML à partir d'un code source. Cette fonctionnalité très utile a notamment servit pour le serveur (en JAVA), les schémas aidant à l'explication et à la réutilisation du projet.



Interface de BOUML

Voici une anecdote concernant BOUML : En septembre 2010, Bruno Pagès, auteur du logiciel, annonce qu'il arrête définitivement de développer BOUML suite à un conflit sur Wikipédia. Une version « ultime » est néanmoins mise en ligne en novembre 2010, suivie par des patches correctifs en 2011.

Suite à la sortie des patches correctifs, l'auteur décide de porter le projet en Qt version 4 et de changer la licence. Le logiciel est disponible en version payante en février 2012. Un

logiciel de visualisation gratuit est disponible.

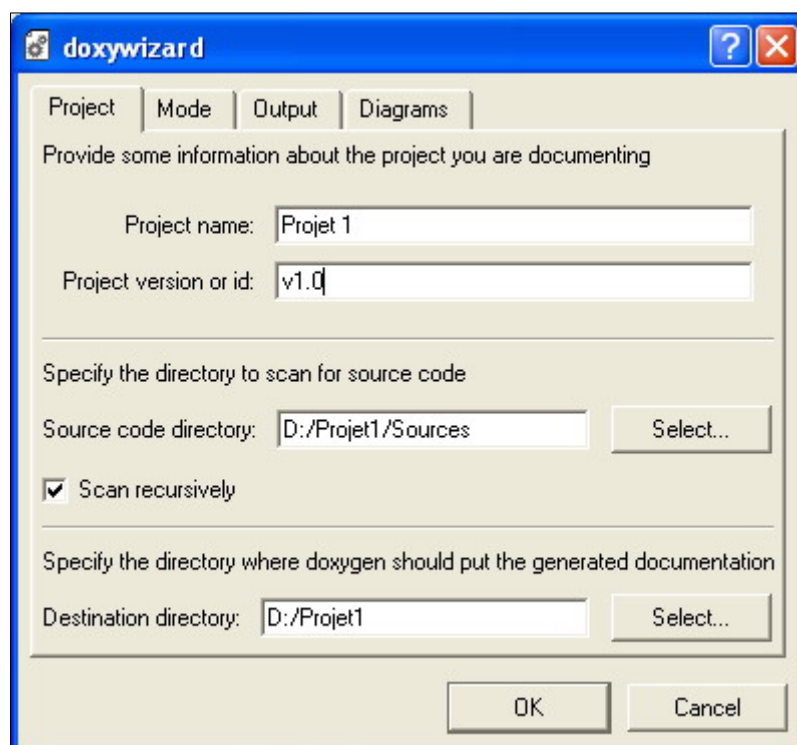
2) Doxygen



Doxygen est un logiciel informatique libre permettant de créer de la documentation à partir du code source d'un programme. Pour cela, il tient compte de la grammaire du langage dans lequel est écrit le code source, ainsi que des commentaires s'ils sont écrits dans un format particulier. Le code de Doxygen a été écrit en grande partie par Dimitri van Heesch.

Doxygen est la contraction de « dox » (« docs », abréviation anglaise de « documents ») et de « gen » (« generator »), « créateur de documentation ».

Doxygen est capable d'analyser des fichiers sources écrits dans les langages C, C++, Java, Objective C, Python, IDL, VHDL et dans une certaine mesure PHP, C#, D et Fortran. La documentation peut être générée dans l'un ou plusieurs des formats suivants : HTML (comprimé ou non), LaTeX, RTF, PostScript, PDF avec hyperliens, et prochainement XML (en cours de développement).



Interface de Doxygen

IV. Analyse

A) Problèmes listés dans le cahier des charges

Le cahier des charges établi lors de la conception du projet divise les fonctionnalités désirées dans le projet en deux catégories : les fonctionnalités obligatoires, qui sont à implémenter en priorité, et les fonctionnalités optionnelles qui sont à implémenter s'il reste du temps.

1) Fonctionnalités obligatoires

➤ Serveur :

- Gestion de parties : Permettre à des clients (multi plate-forme) de se connecter à une partie.
- Gestion du jeu : le serveur doit gérer tout ce qui se passe dans le jeu et indiquer aux différents clients, en temps réels, différents évènements et interactions qui se passent dans la partie.
- Une règle de poker doit pouvoir se dérouler (Tournoi) pour une partie avec une seule table.
- Gestion des clients : le serveur doit prendre en compte les connexions, déconnexions, différents messages des clients, quel que soit le moment où il les reçoit.
- Gestion des données des clients : le serveur doit enregistrer les différents clients qui se connectent, leur mot de passe et leur nombre de parties gagnées.

➤ Clients :

- Connexion au serveur : Les différents clients doivent se connecter et communiquer avec le serveur de jeu.
- Interface : Les clients doivent fournir une interface interactive avec le joueur pour lui permettre de choisir une partie, de faire les choix qui lui sont disponibles pendant une partie et de connaître les informations nécessaires au bon déroulement d'une partie. (Jetons....).
- Pour Android, des fonctionnalités supplémentaires seront nécessaires. Voici la liste des fonctionnalités à ajouter :
 - L'interface doit supporter toutes les tailles d'écrans qui existent actuellement.
 - Au vu des petits écrans, l'interface doit être réfléchie afin de la rendre simple d'utilisation et intuitive.
 - Afin d'économiser la batterie, les ressources du programme devront être minimalistes.

2) Fonctionnalités optionnelles

➤ Serveur :

- Différents types de parties : le serveur peut créer des parties avec des règles de poker différentes.
- Chat : gestion d'un chat entre les joueurs d'une même partie.
- Parties multi tables : gestion de parties multi tables.
- Gestion des jetons améliorée : gestion des jetons des joueurs améliorée, leur permettant d'en récupérer tous les jours etc...
- Gestion des statistiques des joueurs (parties gagnées, plus grosses mises...)

➤ Clients :

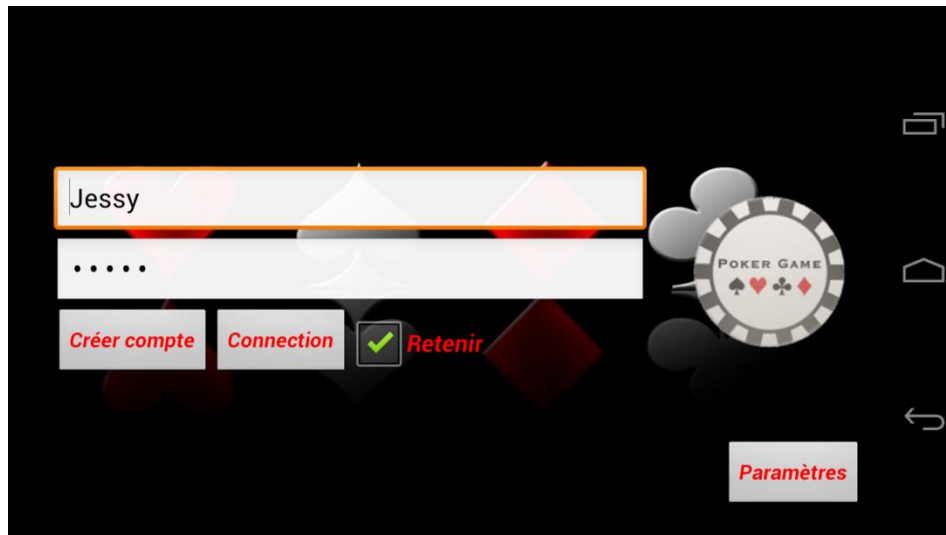
- Interface pour un chat : modification de l'interface pour y mettre un emplacement permettant aux joueurs de chatter.
- Android :
 - Utilisation de la librairie OpenGL es 1.1 afin d'ajouter des animations à l'application.
 - Ajout d'une fenêtre pour consulter ses statistiques et celles des autres joueurs.

B) Aspect fonctionnel de la partie Android

1) Aspect fonctionnel de l'application Android

Notre application possèdera cinq écrans (un écran utilise le principe d'activity propre à Android que nous décrirons plus loin), ce découpage a été fait afin d'améliorer l'ergonomie ainsi que l'expérience utilisateur :

- L'écran d'accueil où l'utilisateur aura la possibilité de se connecter, de créer un compte ou d'accéder aux paramètres de l'application via le bouton paramètre (version d'Android supérieur à 3.0) ou la touche paramètre pour les anciennes versions d'Android.



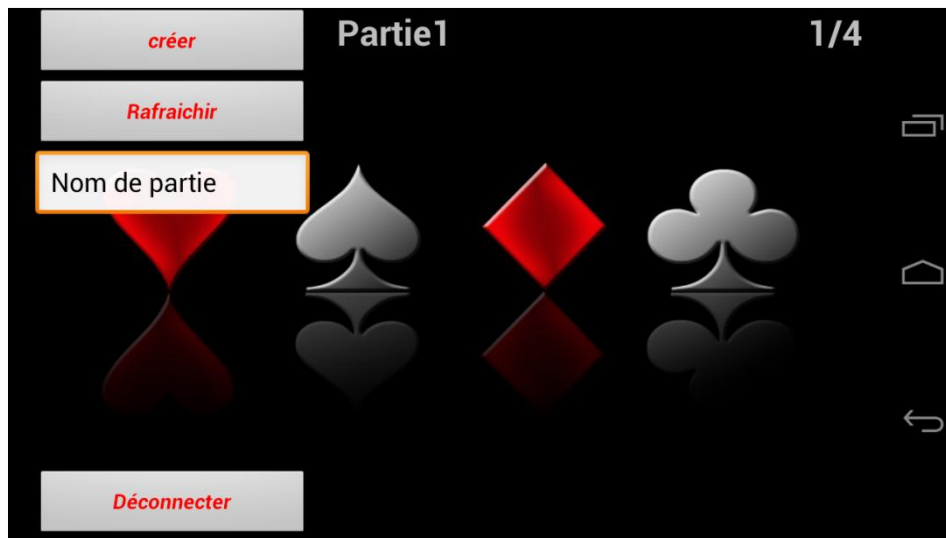
Ecran d'accueil de l'application Android

- L'écran de paramètre où l'utilisateur pourra changer son pseudonyme ou son mot de passe.



Ecran de paramètre de l'application

- L'écran avec la liste des parties. Sur cette écran l'utilisateur pourra soit créer une partie, soit en rejoindre, soit se déconnecter. On affiche dans la liste le nom de la partie et le nombre de joueurs présents dans le salon d'attente.



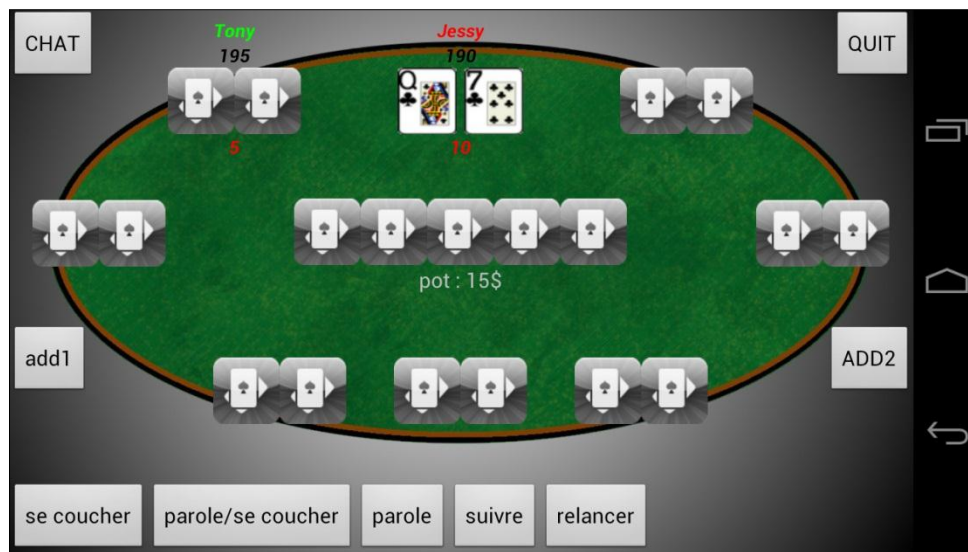
Ecran avec la liste des parties

- L'écran avec la liste des joueurs dans la partie sélectionnée par le joueur. Permet d'avoir la liste des joueurs qui seront dans la partie. La liste s'actualise en temps réel en fonction des gens qui décident de rejoindre ou quitter le salon d'attente. Le créateur de la partie est le premier dans la liste.



Ecran avec la liste des joueurs

- L'écran de jeu qui possèdera deux parties. La première sera la table de jeu avec les cartes et les jetons et la deuxième sera la fenêtre de chat permettant de discuter avec les autres joueurs.



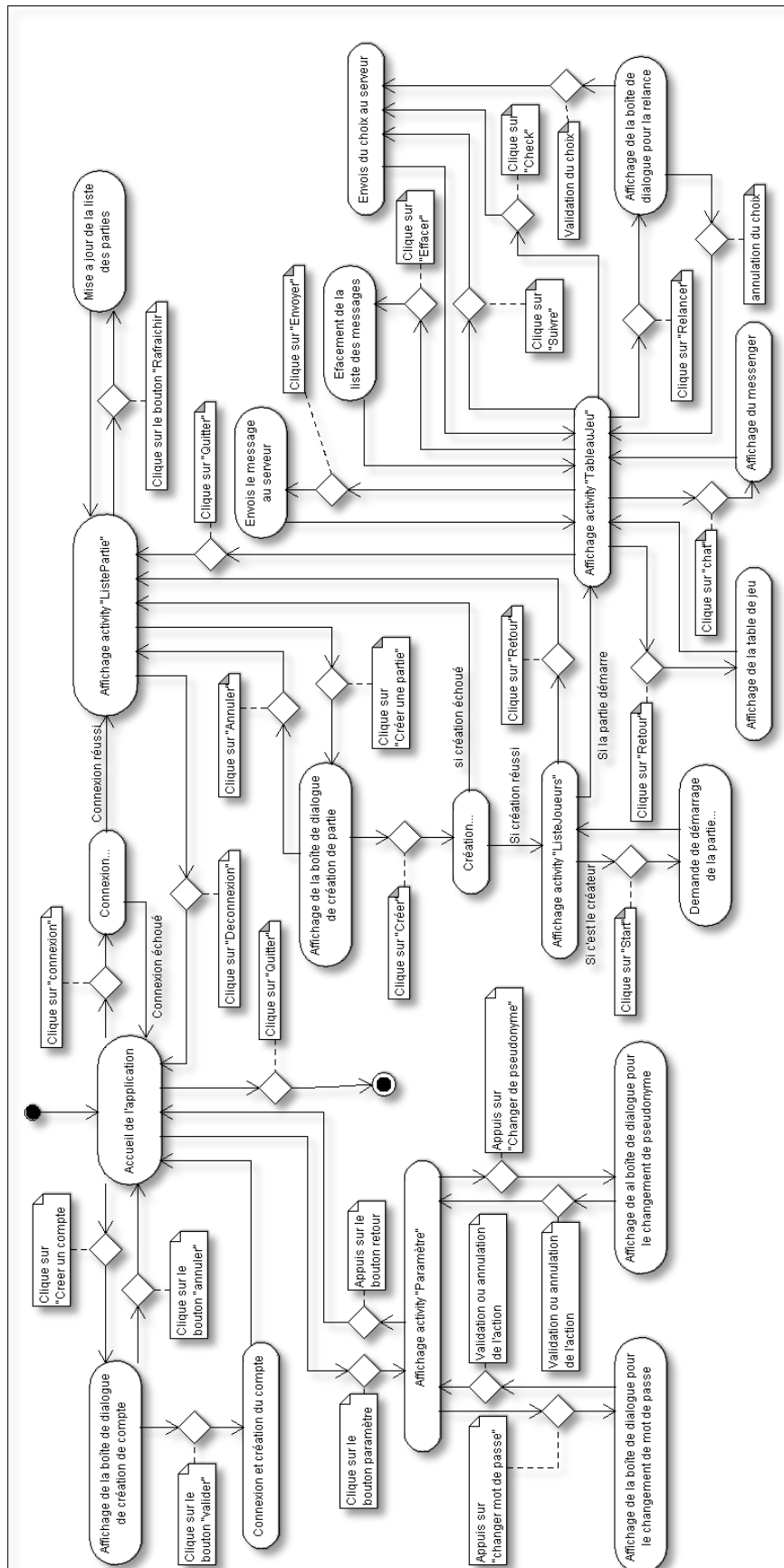
La table de jeu



La fenêtre de chat

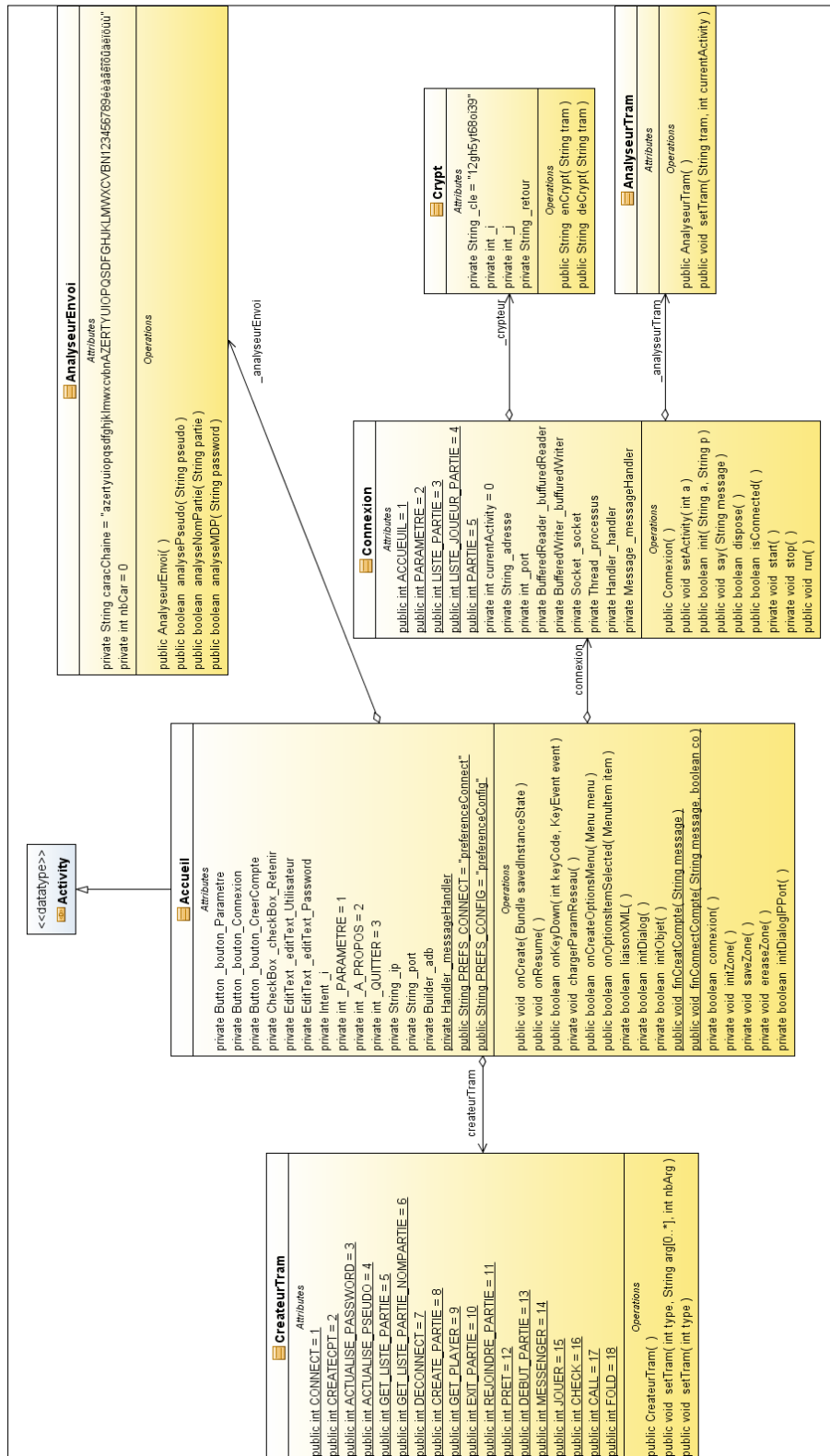
Au démarrage de l'application, l'utilisateur arrivera donc sur l'écran d'accueil sur lequel il pourra accéder aux paramètres, créer un compte ou se connecter. Une fois connecté, il verra s'afficher la liste des parties disponibles. Il pourra également créer une partie personnalisée. Il arrivera ensuite sur un écran d'attente où il verra les joueurs qui veulent rejoindre la partie. Le créateur lancera alors la partie et l'utilisateur arrivera sur la table de jeu et pourra enfin jouer.

Voici donc le schéma navigationnel de notre application :

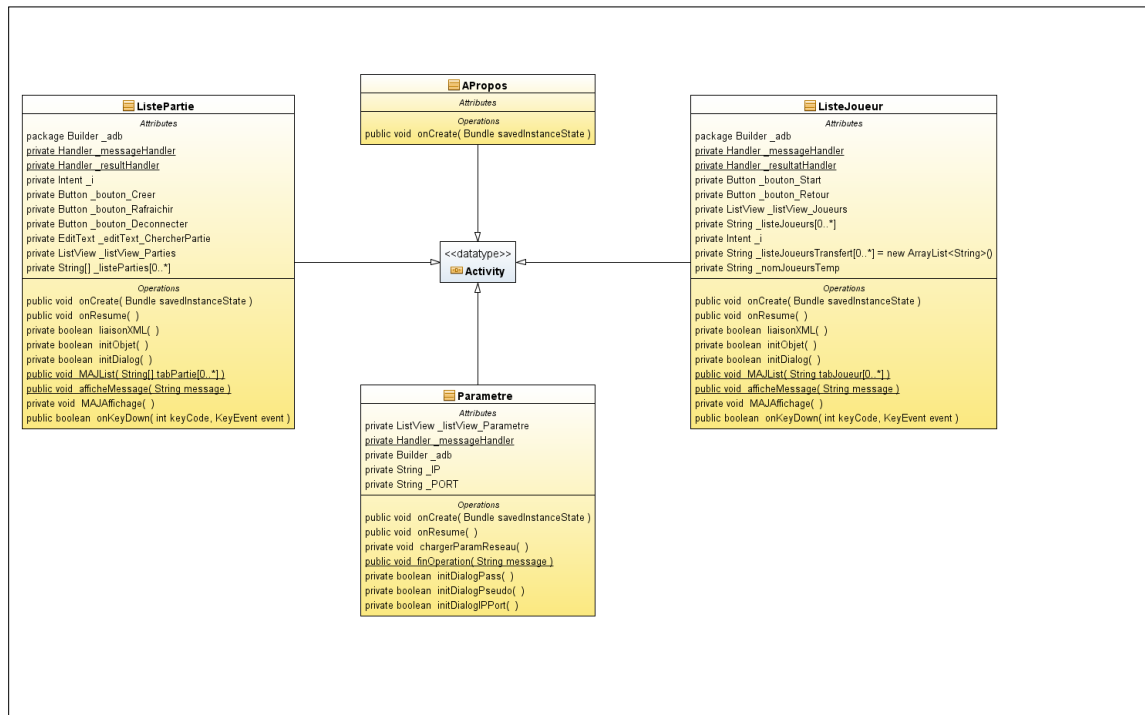


Maintenant, nous pouvons créer le squelette de notre application :

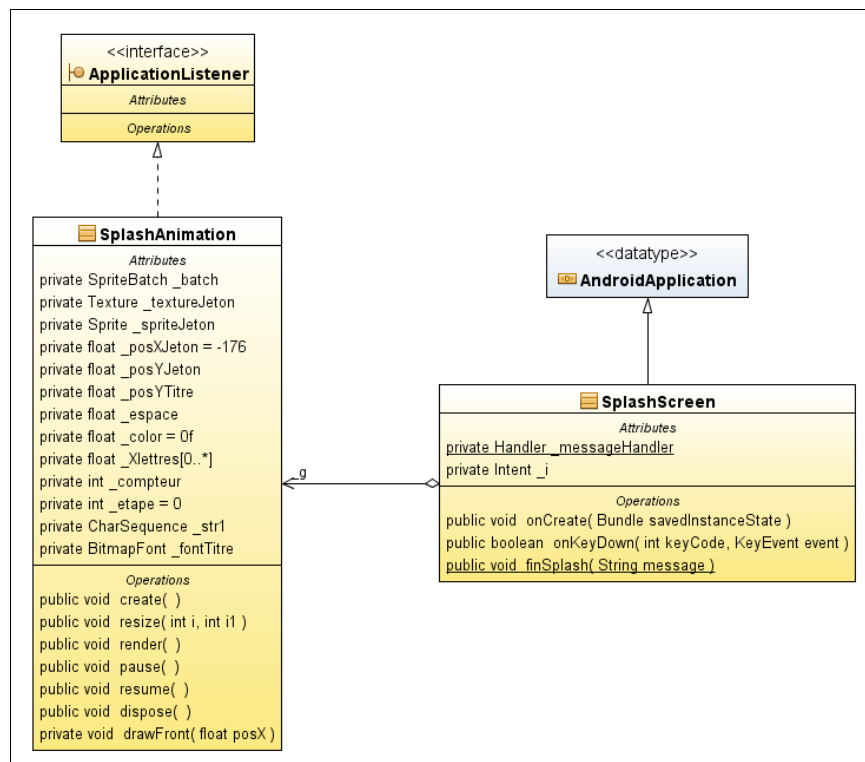
- Ici l'écran principal de notre application : « Accueil ». Elle crée toute les variables nécessaires à la communication telles que la variable de connexion ou l'analyseur de trames.



- Ensuite, nous avons les différents écrans simples : le « APropos », le « Parametre », le « ListePartie » et le « ListeJoueurs ».



- Il y a l'écran de « SplashScreen » qui contiendra l'animation de démarrage de l'application.



- Le dernier écran est celui de la table de jeu : « TableauJeu ». Il contient le système de messenger ainsi que la table de jeu.



Voici donc l'explication sur les différentes classes de l'application :

- « **Connexion** » est la classe qui permet de gérer la socket de connexion ainsi que les flux d'entrée et de sortie.
- « **AnalyseurTram** » est la classe qui permet d'analyser les trames reçues du serveur. La fonction d'analyse est appelée par la classe « Connexion » uniquement et met à jour l'interface en fonction de l'activity courante.
- « **CreateurTram** » est la classe permettant de créer les trames à envoyer au serveur. Les activity appellent la fonction de création et celle-ci crée des trames correctes avant de les passer à la classe Connexion pour l'envoi.
- « **AnalyseurEnvoi** » est la classe permettant de vérifier les pseudonymes, les mots de passes et les noms de partie avant leurs envois au serveur. Cela permet d'éviter un usage intempestif du réseau si l'utilisateur fait des erreurs de format dans l'écriture de ses informations.
- « **Crypt** » permet de crypter et de décrypter les données échangé avec le serveur afin d'améliorer la sécurité des échanges.
- « **SplashScreen** » est la classe qui va permettre d'afficher notre animation au démarrage de l'écran. Elle utilisera la classe « **SplashAnimation** » qui en implémentant les méthodes de la classe « ApplicationListener » permettra de jouer les animations voulues.
- Les autres classes sont les écrans de l'application. Ce sont ces classes qui permettent de faire l'interface homme-machine afin d'utiliser l'application.

2) Programmation de l'application Android

➤ **Le XML :**

Comparé à un programme JAVA normal, Android propose un autre système de construction des Interfaces Utilisateur, se basant sur des fichiers XML. La structure générale des XML layout d'Android est simple : une arborescence d'éléments XML là où chaque nœud porte le nom d'une classe de type View. Naturellement, on peut utiliser le nom de toute sous-classe View et même de celles que nous pourrions créer nous-mêmes. Cette structure nous permet de créer rapidement nos UI et de les maintenir facilement dans le futur. Le plugin Android crée automatiquement un fichier XML layout : main.xml.

Dans notre application nous utiliserons l'objet *AbsoluteLayout*. Dans ce layout, la règle est simple : on définit les positions des objets relativement à leurs voisins et au layout. Donc on va par exemple dire que l'image est plaquée en haut à gauche, tandis que le texte est plaqué en haut du layout, et se trouve à droite de l'image. L'intérêt est que l'on peut donc remplacer tous les *LinearLayout* nécessaires par un seul *AbsoluteLayout*.

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    android:id="@+id/layoutPrincipal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    ...
</AbsoluteLayout>
```

La première partie, appelée prologue permet d'indiquer la version de la norme XML utilisée pour créer le document (cette indication est obligatoire) ainsi que le jeu de caractères (en anglais encoding) utilisé dans le document (attribut facultatif, ici on spécifie qu'il s'agit du jeu utf-8). Ensuite, c'est notre *AbsoluteLayout*. Nous l'ouvrons puis le déclarons grâce aux différents attributs qui suivent :

- Android:id sert à donner un identifiant à notre écran pour qu'il puisse être manipulé par notre code JAVA. Ici, il s'appellera donc « layoutPrincipal ».
- Les deux lignes suivantes, *android:layout_width* et *android:layout_height* servent à définir la taille de l'écran de notre application. Ici, l'attribut « fill_parent » signifie que le *layout* occupera toute la place disponible.
- La dernière ligne quant à elle définit l'espace de nom utilisé par l'application.

Il ne faut également pas oublier de refermer les balises que l'on a ouvertes. Donc ici nous refermons la balise de notre *AbsoluteLayout*.

Dès lors notre interface peut être initialisée mais ne comporte encore aucun objet. Sur Android, les objets de l'interface graphique sont composés d'un type et d'un ensemble de paramètres :

```
<EditText
    android:id="@+id/port"
    android:layout_width="76px"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:layout_x="211px"
    android:layout_y="3px">
</EditText>
```

Ici, nous avons simplement une zone de texte Mais d'autres types existent :

| | |
|-----------------|--|
| TextView | C'est le widget le plus simple. Il est semblable au label des applications JAVA normales. Ce sont des chaînes de textes non modifiables par l'utilisateur. Ils servent généralement à identifier les widgets qui leur sont adjacents (« Nom : » placé à côté d'une zone de saisie. |
| EditText | Ce widget permet avant tout à l'utilisateur de saisir du texte. Il est assez semblable au TextView mais apporte des fonctions en plus tel que la correction automatique ou forcer l'utilisateur à utiliser les chiffres. |

| | |
|------------------|--|
| ImageView | C'est l'équivalent image du TextView vu plus haut. Il possède un attribut supplémentaire appelé <i>android:src</i> servant à donner la source d'une ressource graphique à l'objet. |
| Button | Ce widget permet à l'utilisateur de lancer des actions. Le bouton possède un écouteur et dès que l'utilisateur l'utilise, un code défini dans le programme est exécuté. |
| ListView | Ce widget permet de faire des listes avec presque tout ce que l'on souhaite. Nous avons juste besoin de lui fournir un tableau et Android remplira la liste. Pour des listes plus complètes (avec renvois vers une autre activity par exemple), il faudra coder des écouteurs pour les lignes de la liste. |

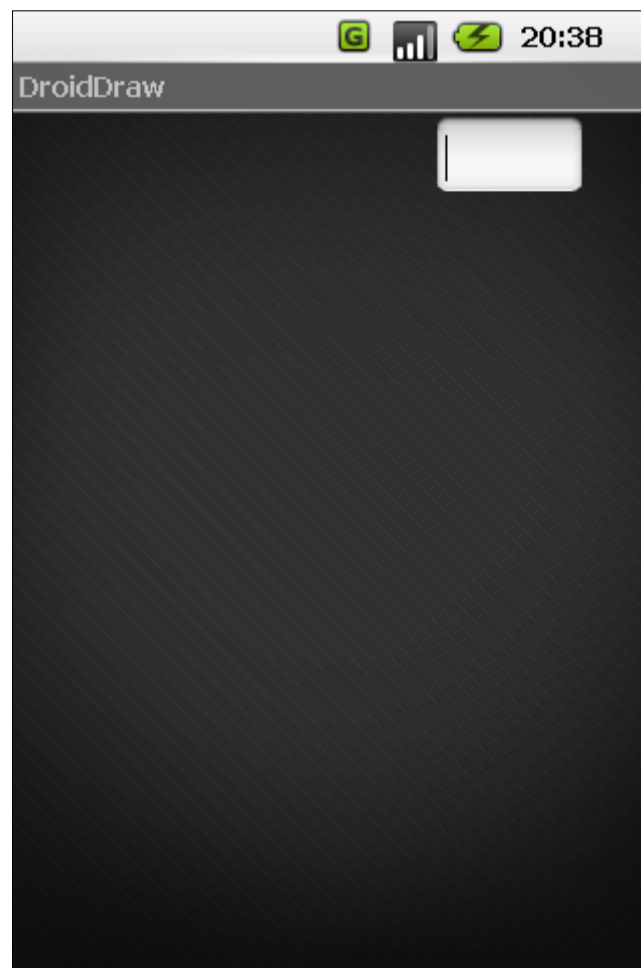
Le code précédent sert donc à initialiser dans notre layout un objet de type EditText. Comme pour l'AbsoluteLayout, l'objet aura une id pour pouvoir être identifié dans le code JAVA et une taille définie par les attributs *android:layout_width* et *android:layout_height*. Il y a ensuite trois nouveaux attributs :

- *android:textSize* permet de définir la taille du texte que l'utilisateur écrira dans la zone de texte. Ici la taille sera de 18.
- Les deux attributs suivant définissent le point d'origine de l'objet par rapport au premier point en haut à gauche du layout dans lequel il se trouve (*android:layout_x* et *android:layout_y*)

La structure finale de ce code XML donnera ceci :

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    android:id="@+id/layoutPrincipal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    ...
    <EditText
        android:id="@+id/port"
        android:layout_width="76px"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:layout_x="211px"
        android:layout_y="3px">
    </EditText>
</AbsoluteLayout>
```

Ce code XML que nous venons de voir en détail donnera cette interface :



Voici notre interface qui possède donc un seul layout de type *AbsoluteLayout*. Notre objet de type *EditText* est positionné à 211 pixel sur l'axe x et 3 pixel sur l'axe y sachant que l'origine du layout se situe en haut à gauche de l'écran. Il a une hauteur qui correspond à la taille du texte qu'il contient. Comme nous avons défini la taille du texte à 18, il est adapté pour afficher du texte de taille 18. La largeur quant à elle est définie à 76 pixels.

Maintenant nos objets sont créés et notre interface est prête. Pour utiliser nos objets dans notre code JAVA, il faut créer une variable JAVA du même type correspondant à l'objet voulu. On commence par créer l'objet de type *EditText* :

```
EditText port;
```

Ensuite on fait pointer notre objet XML vers cette variable :

```
port = (EditText) findViewById(R.id.port);
```

On commence par récupérer l'objet XML grâce à son id. On ajoute une vue sur cet objet grâce à la fonction `findViewById()` que l'on transforme en type `EditText`. Cela nous permet à l'aide de notre variable `EditText` de contrôler l'objet `EditText` de notre interface.

Voilà donc comment doit être construit le fichier `main.xml` qui contiendra tous les objets et détails de l'interface graphique. Notre interface comportera au minimum une liste et un bouton de connexion.

Dans un deuxième temps, on utilise le XML dans le fichier « `AndroidManifest.xml` » qui est le fichier de configuration de l'application. C'est le point de départ de toute application Android. Il se trouve à la racine du projet. C'est dans ce fichier que l'on déclare ce que contiendra l'application (les `activity`, les `services`, etc...). On y indique également la façon dont ces composants seront reliés au système Android lui-même en précisant, par exemple, l'activité qui doit apparaître dans le menu principal du terminal (aussi appelé « lanceur » ou « `launcher` »).

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.me.socket21">
    <application>
        <activity android:icon="@drawable/icon"
            android:name=".MainActivity"
            android:label="Nom Appli">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

Comme pour le XML de l'interface graphique, on commence par définir le type de document. Un XML avec sa version et son encodage. Ensuite nous commençons le fichier *manifest*. Cette balise possède l'espace de nom Android et le package de l'application :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.me.socket21">
```

Après avoir défini le manifeste, nous définissons l'application avec l'*activity*. Nous paramétrons l'application avec trois paramètres. Le premier permet de donner un icône à l'application. Cet icône se trouve dans le répertoire *drawable* de l'application. Il y a ensuite le nom de l'application. Ici on a le nom par défaut donné par l'environnement de développement. Pour finir on a le label. Il correspond au nom que l'utilisateur verra sur son écran lorsqu'il voudra lancer l'application :


```
<activity android:icon="@drawable/icon"
          android:name=".MainActivity"
          android:label="Nom Appli">
```

Nous avons ensuite l'élément fils « intent-filter » qui lui permet de donner des conditions sous lesquelles l'activité s'affichera. La balise « action » annonce que cette activité est l'activité principale de cette application et la balise « category » annonce quant à elle que l'activité appartient à la catégorie « LAUNCHER », c'est-à-dire qu'elle aura une icône dans le menu principal d'Android. Cette activité étant l'activité principale, Android sait qu'elle est le composant qu'il devra lancer lorsqu'un utilisateur choisira cette application à partir du menu principal :

```
<intent-filter>
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
```

Et pour finir il y a les permissions Android. Juste avant la fermeture de la zone « manifest » se trouve la zone contenant les permissions que le téléphone accorde à l'application. Notre exemple n'a pas besoin de permissions mais voici un exemple de permissions permettant à l'application d'utiliser internet :

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Voilà donc comment bien comprendre et bien paramétrer le fichier *AndroidManifest.xml* afin de configurer une application Android correctement.

➤ Les menus :

Android permet la création de menus. Ces menus, qui s'affichent en bas de l'écran lors de l'appuie sur le bouton « menu » et qui proposent alors différentes options, sont particulièrement bien adaptés pour accéder à des commandes occasionnelles. Nous utiliserons donc ce composant dans notre application notamment pour accéder aux paramètres de l'application.

La création d'un menu se fait en deux parties :

- La création d'un fichier XML décrivant notre menu. On utilise pour cela la balise <menu> dans laquelle on placera des balises <item> pour chaque élément qui sera présent dans le menu. Chaque item possèdera comme attribut un titre qui est en fait le texte affiché sur le choix correspondant et un attribut id pour l'implémentation du choix dans le code Java (comme pour tout composant Android).

Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">
  <item
    android:id="@+id/parametre"
    android:title="Paramètres"
  />
  <item
    android:id="@+id/credits"
    android:title="Credits"
  />
</menu>
```

Ce code définit un menu avec deux choix : appelés paramètres et crédits.

- La deuxième partie de la création de menu est l'implémentation Java du menu décrit en XML. Pour cela, il suffit de surcharger la méthode `onOptionsItemSelected` (Menu menu) puis de faire un objet `MenuInflater` dans lequel on appelle la méthode `inflate` avec en paramètre le XML de notre menu.

```
public boolean onOptionsItemSelected(Menu m) {
    new MenuInflater(getApplicationContext()).inflate(R.menu.menuprincipal, m);
    return (super.onOptionsItemSelected(m));
}
```

On obtiendra avec le XML précédent, ce résultat :



Lorsque l'utilisateur sélectionne un choix dans le menu, on utilise la fonction `onOptionsItemSelected` avec en paramètre l'objet `menuItem` choisi. Il nous suffit donc de regarder qu'elle objet est renvoyé et ainsi effectuer les instructions correspondantes :

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (m.getItemId()) {
        case R.id.parametre :
            //Instructions d'affichage de la page de paramètres
            break;
        case R.id.credits :
            //Instructions d'affichages de la page de credits
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

➤ Les préférences :

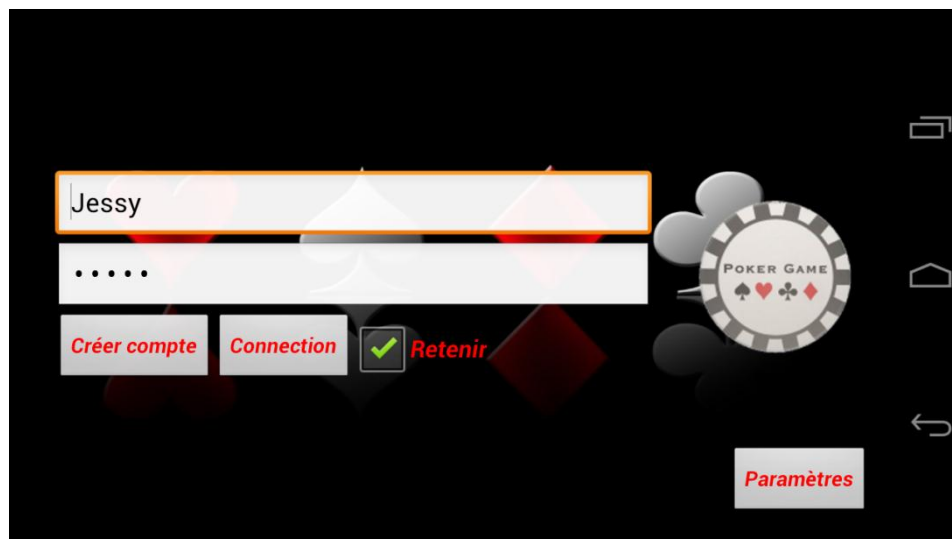
Pour notre application nous utiliserons aussi la notion de préférences. Le système de préférences sous Android est un outil très pratique se présentant plus ou moins comme une base de données. Les préférences permettent d'associer à une valeur de type primitif, une chaîne de caractères appelée généralement clé. Les préférences permettent de stocker, comme leur nom l'indique, des valeurs qui doivent rester fixes au cours du temps (notamment lorsque l'application est terminée puis rouverte) : les données sont persistantes. Les préférences sous Android sont de trois types :

- Les préférences spécifiques accessibles via la fonction `getPreferences`.
- Les préférences communes à toute l'application accessibles via `getSharedPreferences`.
- Les préférences communes à tout le système Android accessibles via `getDefaultSharedPreferences`.

Pour notre application, nous utiliserons *getSharedPreferences*. Ainsi, nous pourrons ensuite éditer les préférences d'Android.

Android fournit en outre des objets XML déjà implémentés pour la modification de préférences, on utilise pour cela un layout particulier, appelé *PreferenceScreen* dans lequel on placera les équivalents de chaque objet pour les préférences comme *EditTextPreference*, *ListPreference* etc...

Au final, ces objets implémentés dans le SDK Google permettent d'automatiser les tâches de modification et de sauvegarde des nouvelles préférences.



Utilisation du système de préférence pour sauvegarder le nom d'utilisateur et le mot de passe

➤ Les activity :

Lors du développement d'une application Android, il faut garder à l'esprit que chaque écran correspond à une activité (il est aussi possible de créer des activités sans interface graphique). Ce découpage qui semble de prime abord contraignant se révèle très efficace pour la sécurité. Chaque activité se déroule dans son propre environnement et ne communique que de manière très restreinte avec les autres activités.

Pour déclarer une activité, il faut utiliser un filtre d'intention. Un filtre d'intention correspond à une instruction donnée telle que « lance depuis cette classe, l'activité présente dans telle classe ». Cela se traduit dans le code par :

```
preferencesIntent = new Intent(this, preferences.class);
```

Ensuite, il suffit de passer l'intent en paramètre à la fonction *startActivityForResult* qui prend aussi un identifiant afin de lui associer l'activité lancée :

```
startActivityForResult(preferencesIntent, 0);
```

Cet identifiant sera renvoyé en paramètre de la fonction *onActivityResult* qui est elle-même appelée lorsqu'une activité se termine, ce qui permettra d'agir en conséquence.

La communication entre deux activités est compliquée, en réalité il est uniquement possible de passer entre deux activités des types primitifs. Or, dans notre application, nous devons partager entre chaque activité notre connexion ainsi que nos buffers. Pour cela, nous avons déclaré comme variable de classe les objets correspondants à ces deux points. La connexion et les buffers étant, de toute manière, uniques dans tout le code. Dans notre projet, nous avons d'ores et déjà identifié les activités suivantes :

- L'activity de l'écran d'accueil.
- L'activity pour afficher l'écran de paramétrage de l'application.
- L'activity pour changer l'état d'un interrupteur.
- L'activity pour acquérir les données d'un capteur.

➤ Cycle de vie d'une application :

Pour finir avec les notions importantes de la programmation Android que nous utilisons dans le projet, il convient d'aborder le cycle de vie d'une activité. En effet, une activité peut passer dans différents états et entre chaque changement d'état, une fonction est appelée automatiquement. La surcharge de ces différentes méthodes nous permettra d'agir au mieux en fonction de ce que l'utilisateur fait en même temps que l'utilisation de l'application.

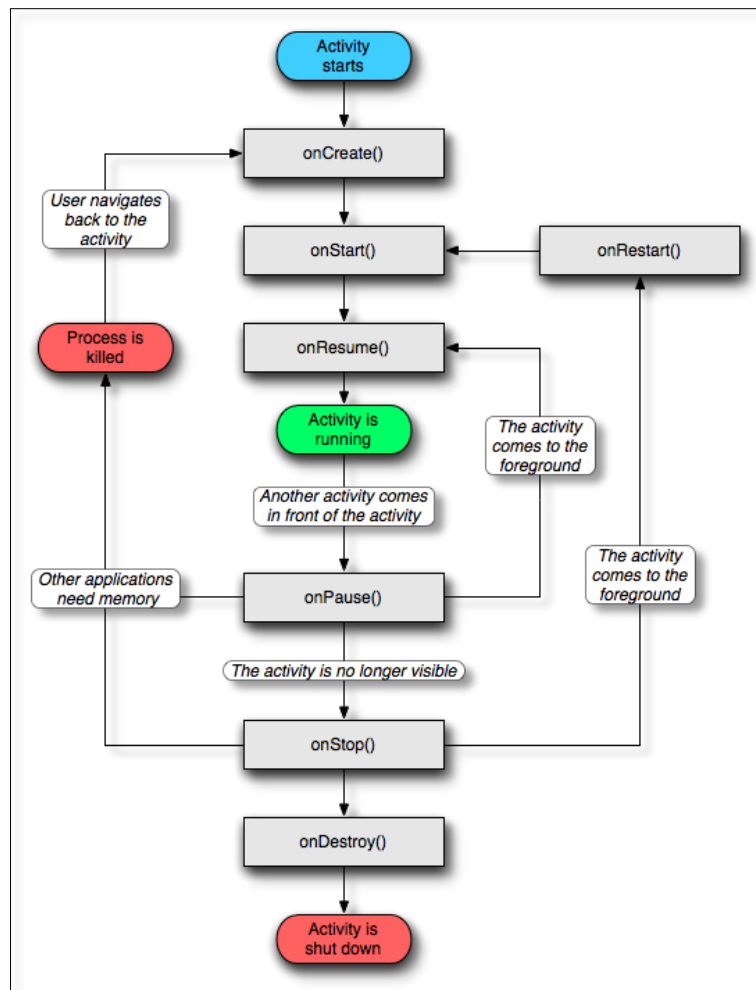


Schéma fournit par Google pour représenter le cycle de vie d'une activity

On peut constater que la méthode `onCreate` est appelée uniquement lors du premier lancement de l'application, c'est ici que l'on devra faire toutes nos initialisations (faire toutes les initialisations dans la méthode `onResume` par exemple consommerait beaucoup plus de ressources car la fonction est appelée plus souvent). Lorsque qu'une information surgit au-dessus de l'activité (lorsque le réveil s'active par exemple), cette dernière est mise en pause et la fonction `onResume` est appelée quand l'application revient au premier plan (il peut être judicieux de désactiver tous les écouteurs à ce moment afin d'éviter de déclencher des événements malencontreusement). Enfin, lorsque l'utilisateur change d'écran, l'activité est stoppée. Lorsque qu'une activité est stoppée, elle peut être détruite par le destructeur Android à tout moment si un besoin de mémoire est formulé par une autre application. Si l'application est détruite, `onDestroy` est appelé, c'est d'ailleurs la dernière fonction avant la disparition totale de l'instance de l'application et au lancement de l'application, `onCreate` sera appelé. Dans le cas où `onStop` ne conduit pas à une destruction, la méthode `onRestart`, `onStart` et `onResume` sont appelés quand l'utilisateur retourne dans l'application. Il sera donc intéressant dans notre projet d'utiliser au mieux ces appels afin d'optimiser notamment la consommation de l'application, que ce soit au niveau de la mémoire ou de la batterie.

3) Communication avec le serveur

Afin de pouvoir communiquer de manière intelligible avec le serveur, nous avons défini un protocole, permettant d'envoyer et de recevoir des données. Ce protocole consiste en l'envoi de chaînes de caractères que le téléphone ainsi que le serveur découpent pour agir en fonction.

Dans cette chaîne, chaque Informations est délimité par un « @ », puis chaque sous informations est défini par un « / ».

- L'émission

L'émission se fait par l'intermédiaire de la classe « CreateurTram ». Lorsqu'une activity désire communiquer avec le serveur, elle appelle la fonction « setTram() » de cette classe. Cette fonction va ensuite créer la tram et l'envoyer au serveur. Il est possible de créer des trams simples avec seulement un mot clé ou complexe avant plusieurs arguments.

- La reception

Lorsque la classe gérant la connexion reçoit une tram, elle l'envoie à la classe « AnalyseurTram ». Cette classe permet de traiter les chaines en fonction de l'activity de destination. Cela permet de ne pas traiter les trams qui seraient reçus en retard. L'analyseur ne traite que les trams qui sont à destination de l'activity courante. Cette classe fait également le découpage des teams et insère ensuite les données dans des tableaux afin de les transmettre aux activity.

C) Aspect fonctionnel de la partie C++

1) Présentation de l'application C++

L'application C++ a pour but d'échanger des données avec le serveur; cette communication permet le déroulement d'une partie de poker. La communication Client C++/ Serveur s'effectue par l'envoi et la réception de sockets.

Cette application est constituée de différentes fonctions qui interagissent entre elles grâce à des *Threads*.

L'ensemble et le fonctionnement de ces fonctions ne sont pas bien compliqués, elles établissent le lien au sens imagé entre le serveur et l'homme.

Le serveur a besoin de mots clés très précis, l'application demande à l'homme sous forme de questions et d'affirmations, le pseudo qu'il veut et son mot de passe.

Une fois que l'application a les informations qu'elle désire, il ne lui reste plus qu'à les envoyer au serveur qui traite les informations reçues pour ensuite dire si tout est bon ou sinon quel est le problème.

On pourrait donc comparer l'application C++ à un intermédiaire, un traducteur qui parlerait le langage de l'homme et celui de la machine.

L'application C++ est développée sous Visual Studio 2008. Disons plutôt qu'une partie est développée sous Visual 2008 et une autre le sera sous Visual 2010.

Dans un premier temps nous avons codé sous Qt car c'est ce qui était demandé, puis nous nous sommes rapidement tournés vers un logiciel que nous connaissions bien (Visual 2008).

Une fois une bonne partie du projet faite, nous avons réalisé qu'il nous serait impossible de continuer sans l'utilisation de *Threads*. Nous avons donc dû passer sous Visual Studio 2010.

2) Programmation de l'application C++

L'application C++ devait être traitée sous Qt mais pour des raisons techniques (pour que le serveur puisse avancer en même temps que les différents clients), j'ai préféré faire un client dit «en mode console» (donc sans interface graphique). Ceci me permettait d'avancer en même temps que le serveur et donc de tester les fonctions que je créais au fur et à mesure avec celui-ci.

Mon binôme devait en parallèle apprendre le Qt et utiliser mes fonctions pour faire une application avec une interface graphique.

Le Client C++ compte deux fonctions principales. Une qui écoute et une qui lit. En fonction de ce qui a été lu, donc envoyé par le serveur, le client va lancer telle ou telle fonction.

La fonction « écoute » est créée à l'aide de *Thread*, ce qui lui permet d'écouter à n'importe quel moment ce que lui dit le serveur et d'agir en conséquence.

Les problèmes rencontrés :

- Qt :

Ce Client n'aura pas d'interface graphique parce que l'apprentissage de la programmation sous Qt s'est avéré plus longue et plus fastidieuse que prévu.

Un autre problème de Qt c'est qu'il faut joindre les fichiers .dll.

- Visual Studio :

Le projet s'est déroulé comme prévu dans le planning prévisionnel jusqu'à ce que l'on bloque sur les *Threads*. On ne peut pas réaliser ce projet entièrement si on ne maîtrise pas complètement le multitâches.

```
1>.\main.cpp(214) : error C2673: 'main' : les fonctions globales n'ont pas de pointeurs 'this'
1>.\main.cpp(214) : error C2227: la partie gauche de '->ThreadStart' doit pointer vers un
type class/struct/union/générique
1>.\main.cpp(214) : error C2065: 'myThreadDelegate' : identificateur non déclaré
1>.\main.cpp(214) : error C2673: 'main' : les fonctions globales n'ont pas de pointeurs 'this'
1>.\main.cpp(214) : error C3350: 'System::Threading::ThreadStart' : un constructeur délégué
attend 2 argument(s)
1>.\main.cpp(215) : error C2673: 'main' : les fonctions globales n'ont pas de pointeurs 'this'
1>.\main.cpp(215) : error C2227: la partie gauche de '->trd' doit pointer vers un type
class/struct/union/générique
1>.\main.cpp(215) : error C2065: 'myThreadDelegate' : identificateur non déclaré
```

3) Le dialogue avec le serveur

Le serveur envoie des messages grâce à des *sockets* et le Client C++ dispose d'une fonction nommée « écoute » qui a pour but de récupérer ces messages, de les traiter et de réagir en conséquence.

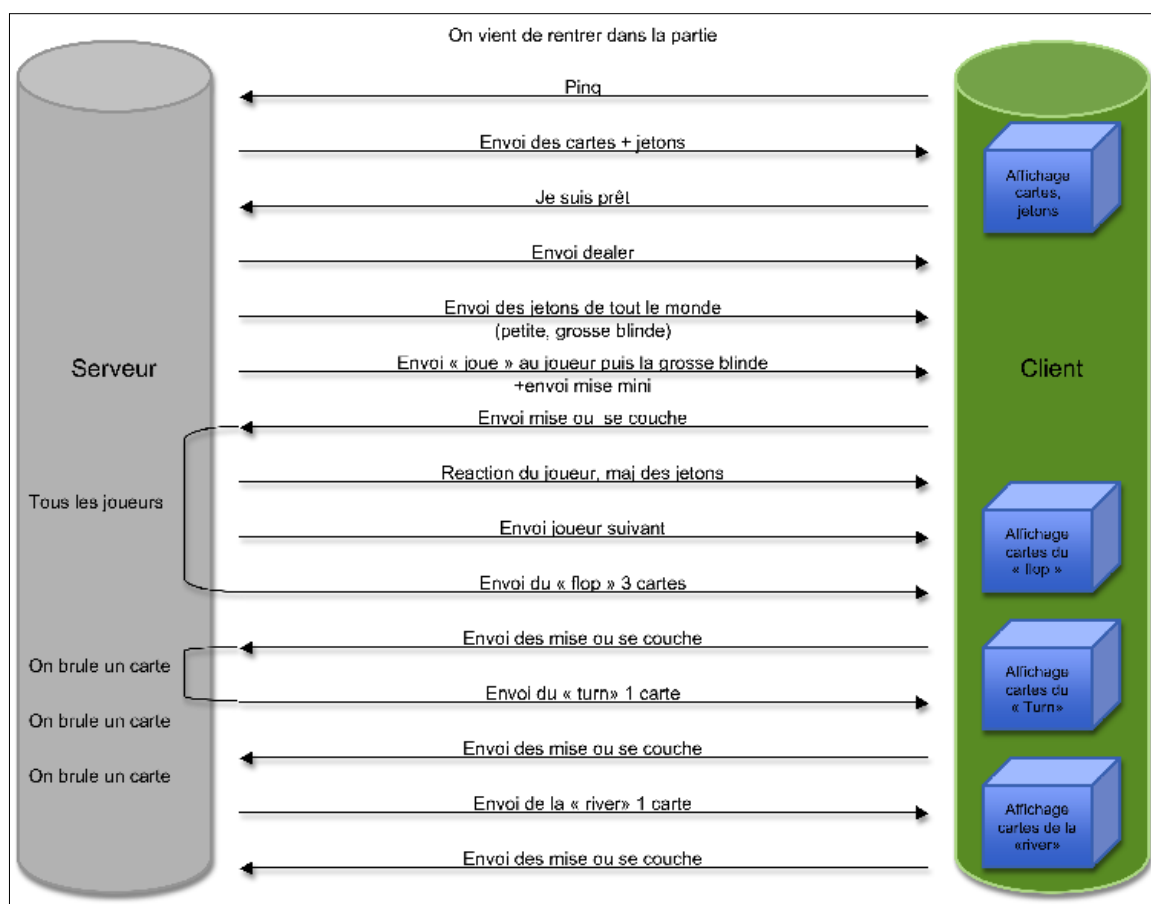
A son tour, le Client C++ envoie des messages qu'il place dans des *sockets*. Ces messages sont traités par le serveur qui retourne un message. Par exemple :

Si on envoie : CREERCOMPTE@NOMDUCOMPTE@PASSWORD.

Le serveur traite l'information.

Et renvoie : OK.

Il y a un protocole très précis qui a été mis en place pour une bonne communication entre le serveur et les clients.



Communication avec le serveur

D) Aspect fonctionnel du client Web

Le client web est composé d'une page HTML, d'un fichier de style, d'une applet Java gérant les sockets et de divers scripts, écrits en JavaScript, gérant la communication avec le serveur et l'affichage.

1) Aspect fonctionnel de l'application Web

- **HTML et CSS :**

Tous les éléments de l'interface tels que les formulaires, tableaux, menus ou le chat sont définis à la suite dans le fichier « index.html ». Leur position, leur taille, leur mise en forme et leur style sont défini quant à eux dans le fichier « style.css ».

Sans les scripts, tous ces éléments seraient affichés en même temps de manière anarchique. Ce sont eux qui gèrent l'affichage dynamique des éléments en fonctions des actions de l'utilisateur.

- **L'applet Java :**

Lors de mes recherches pour établir une communication via socket avec le serveur en JavaScript, je n'ai pas trouvé de solution satisfaisante ou du moins je n'ai pas réussi à les mettre en œuvre. C'est pourquoi je me suis tourné vers une applet Java.

Son principe est simple, la gestion des sockets à proprement parler est faite en Java et la communication entre les deux langages est faite via la page HTML. En Java, on accède à la page grâce à l'instruction « `browser=JSObject.getWindow(this);` ». Pour JavaScript, l'accès à l'applet se fait grâce à « `document.getElementById('JavaSocketBridge');` »

Par exemple lors de la réception d'un message par l'applet, l'instruction suivante est exécutée : « `browser .call("on_socket_get", arguments);` » ce qui a pour effet d'appeler la méthode « `on_socket_get(message)` » du script.

Autre exemple, lors de l'envoi d'un message, l'instruction est exécutée : « `document.getElementById('JavaSocketBridge').send(message);` ». Ainsi la méthode « `send()` » de l'applet est appelée.

Toute la documentation de ce pont entre Java et JavaScript se trouve sur cette page : <http://stephengware.com/projects/javasocketbridge/>

- **La communication avec le serveur :**

L'ensemble des fonctions de communications sont séparées en trois fichiers « `emission_compte.js` », « `emission_partie.js` » et « `reception.js` ». Celles-ci pourraient se trouver dans un seul et même fichier mais je les ai séparées dans un souci de clarté.

Dans « `emission_compte` », est gérée la vérification des champs remplis par l'utilisateur, et, dans le cas où ils sont validés, l'envoi des messages correspondants au serveur. Ces messages concernent uniquement la gestion du compte (connexion, création, changement de pseudo et de mot de passe) autrement dit, le mode non connecté.

Le script « `emission_partie` » a le même rôle, mise à part le fait qu'il envoi des messages relatifs au partie (création de partie, demande de liste de partie, chat, demande d'informations sur un joueur...).

La gestion des messages reçus depuis le serveur se fait dans « reception ». Ceux-ci sont triés en fonction des messages précédemment envoyés et les fonctions correspondantes sont appelées.

- **Gestion de l’affichage**

Toutes les fonctions de modification de l’affichage sont contenues dans le fichier « gestion.js ». C’est-à-dire qu’en fonction des messages reçus depuis le serveur on affiche les éléments HTML, tels que les formulaires, la liste des parties ou les menus. Par exemple la réception du message « CONNECTOK » entraîne l’effacement du formulaire de connexion, et le remplacement du menu hors connexion par le menu en mode connecté.

- **Affichage d’une partie**

Celui-ci est géré par le script « affichage_partie.js » où tous les éléments HTML ainsi que leur style sont créés. La programmation de ce script est orienté objet. Lors de la création d’une partie, un objet *Partie* est instancié et lui-même va contenir des objets *Joueur*, *Cartes* ou encore *Mise*.

2) Programmation de l’application Web

La programmation de tout le client sauf la partie de l’affichage d’une partie de poker a été simplifiée par l’utilisation de JQuery. JQuery est un Framework JavaScript libre qui porte sur l’interaction entre JavaScript (comprenant l’AJAX) et HTML, et a pour but de simplifier des commandes communes de JavaScript. Ainsi, pour donner un exemple, une instruction du type « document.getElementById(‘element’) » est remplacée par « \$(‘#element’) ».

- **Initialisation**

A l’ouverture de la page la fonction suivante est appelée :

```
$(function() {  
    tout_cacher();  
    $(".connecte").hide(); //cacher tous les elements de la classe connecte  
    $("#gestionjeu").hide();  
    $("#accueil").hide(); //cacher l'element dont l'id est accueil  
    $("#boutons").hide();  
    $("#commencer").hide();  
});
```

Fonction de connexion

Avec JQuery, le dollar placé en-dehors d’une fonction revient à utiliser l’évènement « onDocumentReady ». Ainsi, dès que la page est chargée, les instructions ci-dessus sont exécutées. Elles permettent de ne pas afficher les éléments HTML qui ne corresponde pas au mode non connecté. Vous pouvez remarquer la puissance de JQuery, qui avec une petite instruction, cacher un élément.

- **Détection et gestion d'un évènement**

La plus grande partie de ces évènements est déclenchée par des éléments HTML. Ils sont programmés lors de la création d'une balise HTML, comme dans l'exemple qui suit.

```
<form id="creerpartie" class="cache formtaille" onSubmit="javascript:creer_partie(this.nompartie, this.nbjoueurs); return false;">
  <legend>Créer une Partie</legend>
  <label for="nompartie" class="inline" id="labnompartie">Nom Partie : </label>
  <input type="text" name="nompartie" id="nompartie">
  <label for="nbjoueurs" class="inline" id="labnbjoueurs">Joueurs Maximun :</label>
  <input type="text" name="nbjoueurs" id="nbjoueurs">
  <input type="submit" value="Créer partie">
  <input type="button" value="Annuler" onclick="javascript:annuler('creerpartie');">
</form>
```

Gestion d'un évènement

On observe ici la création d'un formulaire de création de partie avec un bouton de validation et d'annulation. Lorsque le bouton dont le type est « submit » est actionné, la méthode *creer_partie* est appelée. La méthode *annuler* est appelée lors d'un clic sur le bouton correspondant.

```
function annuler(id){
    effform(id);
    $("#"+id).hide("slow");
}

function creer_partie(nom, nb){
    resetcreerprt();
    var reg=new RegExp("[2-8]{1}$","g");

    if (nom.value==''){
        $("#labnompartie").text("/!\ entrez un nom.");
        nom.focus();
    }

    else if (nb.value==''){
        $("#labnbjoueurs").text("/!\ entrez un nombre de joueurs.");
        nb.focus();
    }

    else if(!reg.test(nb.value)){
        $("#labnbjoueurs").text("/!\ entrez un nombre entre 2 et 8.");
        nb.focus();
    }

    else{
        set_nom_partie(nom);
        set_createur(true);
        set_reception("EM_PRT", "CREATEPARTIE");
        socket_send("CREATEPARTIE@"+nom.value+"@"+nb.value);
    }
}
```

Action des boutons

La méthode *annuler* consiste à simplement à réinitialiser et cacher l'élément passé en paramètre.

La méthode *créer_partie* contrôle la validité des champs remplis par l'utilisateur et notifie celui-ci en cas d'erreur. Si les champs sont valides, il règle le domaine de réception pour la redirection des messages reçus et envoie le message correspondant à l'action effectuée.

Un autre type d'évènements sont ceux déclenchés par la réception d'un message via le socket. Ce message est analysé et la fonction correspondante est lancée. Voici un morceau de la fonction qui trie les messages :

```
function traiter_message(message) {
    console.log("RECEPTION : " + message + " " + domaine + " " + action);
    if(message=="ERROR") alert("Erreur de syntaxe");
    else if(domaine=="EM_CMPT") {
        if(action=="GETINFO") {
            getinfo(message);
        }
        else if(action=="CONNECT") {
            connect(message);
        }
        else if(action=="ACTPASS") {
            actpass(message);
        }
        else if(action=="ACTPSEUDO") {
            actpseudo(message);
        }
        else if(action=="CREATCPT") {
            creatcpt(message);
        }
    }
    else if(domaine=="EM_PRT") {
        if(action=="CREATEPARTIE") {
            createpartie(message);
        }
        else if(action=="GETLISTEPARTIE") {
            getlistepartie(message);
        }
    }
}
```

Gestion des messages

Celle-ci appelle une méthode qui traitera le message en fonction de l'action effectuée précédemment. On tri d'abord en fonction du domaine (EM_CMPT, EM_PRT, PARTIE) puis en fonction de l'action effectuée dans ce domaine.

Par exemple lors de la demande de la liste des parties, le domaine est réglé sur « EM_PRT » et l'action sur « GETLISTEPARTIE », ainsi le message sera traité par la méthode *getlistepartie* qui se présente comme suit :

```
function getlistepartie(message) {  
    if(message=="NCON")  
        alert("Vous devez être connecté pour voir la liste des parties en cours");  
    else affiche_liste(message);  
}
```

Gestion des parties

Le message est alors comparé aux messages possibles listés dans le protocole. Si le message correspond à la validation de l'action par le serveur, le message est alors traité à proprement parlé. Prenons *affiche_liste* comme exemple :

```
function affiche_liste(message) {  
    var html='';  
    var liste = message.split("@");  
    var partie;  
  
    html += "<TR id='titretable'>" + $("#titretable").html() + "</TR>";  
    for(i=1;i<liste.length;i++){  
        partie = liste[i].split("/");  
        html += '<TR><TD class="ligne_partie" align="center"><a style="display : block; \'  
        text-decoration : none;" href="javascript:rejoindre_partie(\'' + partie[0] + '\')>' + partie[0] + '</a></TD>'  
        for(j=1;j<partie.length;j++){  
            html += '<TD class="ligne_partie" align="center">' + partie[j] + '</TD>';  
        }  
        html += '</TR>';  
    }  
  
    $("#tableparties").text("");  
    $("#tableparties").append(html);  
    $("#tableau").show("slow");  
}
```

Fonction affiche_liste()

Cette fonction traite un message de la forme :

SETLISTEPARTIE@partie/nb joueurs/joueurs max@partie/nb joueurs/joueurs max

Le message est découpé selon les « @ » dans un premier tableau. On boucle sur le tableau. A chaque tour, on découpe la chaîne contenue dans le tableau selon les « / » que l'on place dans un nouveau tableau. Puis on ajoute du code HTML, complété avec les informations contenues dans notre tableau, dans une chaîne de caractère.

Enfin, ce code est inséré dans notre page grâce à la simple instruction *\$("#tableparties").append(html);* générant ainsi un affichage dynamique.

- **Gestion d'une partie**

Pour la programmation de l'affichage d'une partie nous avons essayé de nous rapprocher au maximum d'une programmation objet de classes notamment en rendant les attributs inaccessibles hors de l'objet et en mettant en place des méthodes pour manipuler les objets :

Voici notamment comment déclarer un constructeur d'objet, ici l'objet Div :

```
function Div(id,height,width,top,left){☞};
```

Ici les variables précédés de « var » sont inaccessibles hors de l'objet. Seul des getters et setters permettraient d'y accéder. Les variables *this.siege* et *this.num* sont public donc directement accessibles et modifiables hors de l'objet. Par exemple, si « C » était une instance de *new Carte()* on pourrait faire *C.siege=10* mais pas *C.compteur=10*.

```
function Carte(pos,siege){
  //----- Attributs -----
  var ref=this;
  var ot,ol,t,l,dt,dl,left,compteur,compteur2;
  var retourne;
  var div;
  var carte;

  this.siege;
  this.num;
```

Voici un exemple de méthode :

```
/**
 * @function
 * obtenir L'indice du joueur sur la table en fonction de son pseudo
 * @param {String} pseudo
 * @returns {int} indice
 */
this.getIndiceJoueur=function(pseudo){
  for(var i=0;i<8;i++){
    if(joueurs[i]==pseudo){
      return i;
    }
  }
  return -1;
};
```

Voici comment appeler une méthode :

```
mise.initialiser();
```

Pour simplifier la manipulation de la page html nous avons créé un objet Div ayant en attribut une variable stockant un élément <div> que l'on veut ajouter à la page html. Ainsi il suffit d'instancier un objet Div et d'appeler les différentes méthodes « créer » pour interagir avec le contenu ou l'affichage de la balise. Il ne restera plus qu'à l'ajouter à la page en ajoutant en tant que fils d'une Div déjà existante ou en utilisant la fonction « document.appendChild(div) ».

Ci-dessous une méthode de l'objet Div permettant de modifier son arrière-plan. Elle éclairci le code, permet de changer facilement et rapidement les caractéristiques de l'arrière-plan.

```
/**
 * @function
 * changer l'arrière plan de la div
 * @param {string} l'adresse de l'image à mettre en arrière plan
 * @param {string} par exp "center" ou "top"...
 * @param {string} par exp "repeat" ou "no-repeat"
 * @param {string} la couleur de l'arrière plan (exp "red" ou "#100242" ou "RGB(255,0,142)")
 */
this.setBackground=function(url,position,repeat,color){
    div.style.backgroundImage= "url("+url+")";
    div.style.backgroundRepeat=repeat;
    div.style.backgroundPosition= position;
    div.style.backgroundColor= color;
};
```

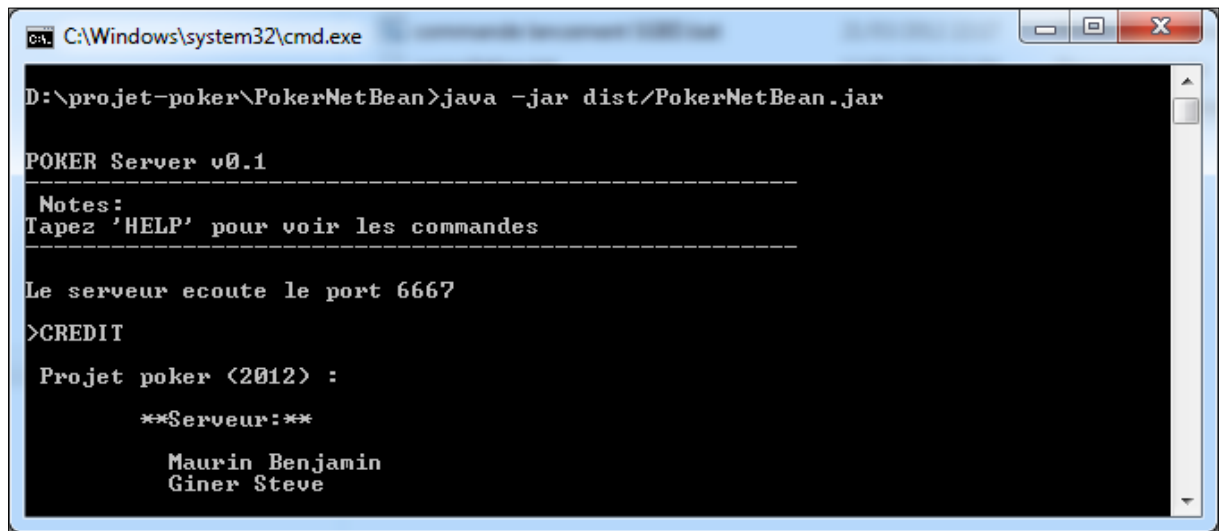
Un des aspects pratiques de JavaScript est la facilité à mettre en des animations peu complexes. Il suffit de mettre un place deux variables ayant comme valeur de départ la position horizontal et verticale de l'objet, puis d'incrémenter/décrémenter ces variables selon un certain intervalle de temps et de donner à l'objet ses nouvelles positions. Voici un exemple de méthode créant une animation :

```
/**
 * @function
 * animation de la carte qui va du centre de la table vers le joueur
 */
this.donner=function(){
    if(div!=null && compteur<100){
        t-=dt;
        l-=dl;
        div.setTop(t);
        div.setLeft(l);
        compteur++;
        compteur2++;
        if(compteur2%10==0){
            if(carte.height<40) carte.height+=4;
            if(carte.width<20) carte.width+=2;
        }
        setTimeout(function(){ref.donner();},1);
    }
};
```

Ici la méthode « donner() » influe sur la position de la variable *div*, qui est une instance du prototype Div. Les variable *t* et *l* représentent la position de *div*. On les décrémente puis on affecte à *div* sa nouvelle position. Puis on rappelle dans un intervalle de temps établi à 1 milliseconde la méthode « donner() » sur l'objet. En plus de la position ici la méthode agit aussi sur la taille d'une image (effet d'agrandissement).

E) Aspect fonctionnel de la partie serveur

Le Serveur contient onze classes pour environ 3700 lignes de code. Il doit gérer la communication et la gestion des clients, le jeu et les parties, la base de données, et le lecteur de commandes.



```
C:\Windows\system32\cmd.exe
D:\projet-poker\PokerNetBean>java -jar dist/PokerNetBean.jar

POKER Server v0.1
-----
Notes:
Tapez 'HELP' pour voir les commandes
-----
Le serveur ecoute le port 6667
>CREDIT
Projet poker <2012> :
  **Serveur:**
    Maurin Benjamin
    Giner Steve
```

Interface du serveur

1) Communication et gestion des clients

Le serveur, lorsqu'il est lancé, écoute indéfiniment les demandes de connections (via l'objet Socket) des clients via sa classe PokerServer (classe main). Lorsqu'il en réceptionne une, il instancie un nouveau *thread* (classe PokerClientThread) et conserve un pointeur vers cette classe. Lorsque la connexion s'arrête, le *thread* est terminé et le pointeur supprimé. Voici le code de la boucle qui écoute les demandes de connexion :

```
while (listening){
    try{
        tempClient = new PokerClientThread(serverSocket.accept());
        clientList.add(tempClient);
        tempThread = new Thread(tempClient);
        tempThread.start();
        screenOut.println("Nouveau client");
    }
    catch (IOException e){
        e.printStackTrace();
    }
}
serverSocket.close();
```

Boucle d'écoute du serveur

Une fois le client connecté, c'est le *thread* créé qui s'occupe d'écouter tous les messages envoyés par la socket du client au serveur. La classe `PokerClientThread` contient une fonction « traitements » qui découpe les messages reçus et réagit en fonction. Voici un morceau du code de cette fonction :

```
/**
 * Fonction qui traite le message en fonction de ses balises
 * @author benjamin Maurin
 * @param inputLine : le message à traiter
 * @param *1 : message traite *0: type de message inconnu *-1 : erreur dans le traitement
 */
int traitements(String inputLine){
    try{
        screenOut.println("Socket recue : "+inputLine+"\n");

        StringTokenizer st = new StringTokenizer(inputLine,"@");
        String cmd = st.nextToken();
        String test1 = "";

        //Demande de connection
        if(cmd.equals("CONNECT")){
            screenOut.println("demande de connection\n");
            test1=st.nextToken();
            PokerServer.testJconnect(test1);
            cmd=PokerServer.bd.verifPassword(test1,st.nextToken());
            send(cmd);
            if(!cmd.equals("CONNECTOK")){
                lecture=false;
            }
            else {connecte = true;pseudo = test1;}
            return 1;
        }
        //Demandes des parties en cours
        if(cmd.equals("GETLISTEPARTIE")){
            screenOut.println("Liste des parties envoyes a un joueur\n");
            if(connecte)
                send(PokerServer.listClientParties());
            else {send("NCON"); }
            return 1;
        }
    }
}
```

Traitement des trams

Les messages reçus sont traités en fonction du protocole (en annexe). Cependant, pour éviter qu'un client mal intentionné ou mal conçu fasse boguer le serveur, de nombreuses vérifications sont effectuées pour que seules les actions possibles pour chaque client soient effectuées.

Par exemple, un client qui n'est pas loggé ne peut pas créer ou rejoindre une partie, un client qui n'est pas le créateur d'une partie ne peut pas la lancer, ect...

De plus, dans le cas de messages erronés ou qui ne font pas partit du protocole, des exceptions sont attendus dans de nombreux endroits critiques du code (à l'aide de try/catch).

La classe `PokerServer` est statique. Ainsi, tous les *threads* peuvent accéder à ses fonctions pour par exemple, recevoir la liste des parties qu'un client demande.

Cette classe est aussi celle qui crée des objets de type `PokerPartie` lorsqu'un client décide d'en créer une. Ces parties sont conservées dans un vecteur (`partiesList`).

Pour faciliter la communication, les objets `PokerPartie` contiennent une liste de pointeur vers les classes/*thread* instanciés des clients qui sont dans la partie.

Comme plusieurs threads peuvent, par l'intermédiaire de PokerServer, créer des parties, modifier la base de données, modifier la liste des clients (déconnection), il faut gérer l'exclusion mutuelle. Ainsi des sémaphores sont utilisés pour éviter les erreurs, tel que la double suppression de parties.

Ici un exemple pour la fonction qui permet de détruire une partie :

```
/**
 * Supprime proprement une partie
 * @author benjamin Maurin
 * @param deadPartie
 */
public static void deletePartie(PokerPartie deadPartie){
    try{
        availableP.acquire();
        int num;
        num=partiesList.indexOf(deadPartie);
        if(num!=-1){
            deadPartie = null;
            rtim.gc(); // appel au garbage collector
            partiesList.remove(num);
            screenOut.println("Une partie est detruite");

        }catch(Exception e){
            e.printStackTrace();
        }
        finally {
            availableP.release();
        }
    }
}
```

Destruction de partie

2) La gestion des jeux

L'ensemble de classes décrites ici doit prendre en charge les évènements du jeu côté serveur comme la donne, l'attente de la réponse d'un joueur, la mise des jetons, etc... Quatre classes sont utilisées pour le déroulement d'une partie :

- PokerPartie : classe principale qui gère le déroulement de la partie (enchères, répartition des jetons, ...).
- Jeu : classe permettant de gérer tout ce qui touche aux cartes (paquet, comparaison des mains, ...).
- PokerClientThread : classe contenant les paramètres du client (jetons, jetons posés, état, ...) et qui permet la communication.
- TimerChoixJoueur : classe permettant d'attendre le choix du joueur pendant 15 secondes.

La partie commence en lançant « `tournoi()` » qui lancera « `déroulement()` » avec en paramètre l'indice du clients qui doit poser la blind dans la liste « `clientList` ».

La liste « clientList » est composée d'instances de « PokerClientThread ». Chaque instance représente un client. Cette classe contient différents paramètres, voici ceux qui nous intéresseront pour le déroulement d'une partie :

- « int jetons[2] » : avec en 0 les jetons du joueur et en 1 les jetons posés.
- « int cartes[2] » : avec les 2 cartes de la main du joueur.
- « int attente » : pour savoir l'état du joueur (0 : en jeu, 1 : couché, 2 : tapis, -1 : perdu).
- « int joue » : pour savoir si c'est son tour (0 : non, 1 : oui).
- « int pot » : jetons que le joueur n'a pas le droit de toucher.

Bien sûr, ce sont des variables de classes privées et ne sont modifiables et utilisables que par des fonctions.

La classe « PokerPartie » est composée de différentes fonctions pour permettre l'envoi des différents messages du protocole utiles au déroulement de la partie.

Au début, de « tournoi() », on instancie la classe Jeu pour avoir le paquet de cartes. Dans Jeu, le paquet de cartes est représenté par une pile d'entier contenant des entiers de 0 à 51 ('tas'). Chaque entier représente une carte différente. Chaque carte a une valeur et une couleur qui sont des entiers allant respectivement de 2 à 14 et 0 à 3. La valeur et la couleur des cartes sont très rapide d'accès car ses données sont présente dans un tableau d'entiers à deux dimensions « vc » de 52 entrées. Si on veut la couleur de la carte 12 on a donc juste à faire « vc[12][1] » et on obtient la couleur. De plus, ce tableau n'est initialisé qu'une seule fois par tournoi puisqu'il reste le même et a ce coût est donc négligeable. Des fonctions sont utilisées dans le code pour permettre tout changement rapide de ce système tout en conservant le code des fonctions.

Nous avons donc des cartes que l'on peut représenter comme ceci :

| couleur | valeur | numéro | nom |
|---------|--------|--------|----------------|
| 0 | 14 | 0 | AS de pique |
| 1 | 14 | 1 | AS de cœur |
| 2 | 14 | 2 | AS de trefle |
| 3 | 14 | 3 | AS de carreaux |
| 0 | 2 | 4 | 2 de pique |
| 1 | 2 | 5 | 2 de cœur |
| 2 | 2 | 6 | 2 de trefle |
| 3 | 2 | 7 | 2 de carreaux |
| 0 | 3 | 8 | 3 de pique |
| 1 | 3 | 9 | 3 de cœur |
| 2 | 3 | 10 | 3 de trefle |
| 3 | 3 | 11 | 3 de carreaux |
| 0 | 4 | 12 | 4 de pique |
| 1 | 4 | 13 | 4 de cœur |
| 2 | 4 | 14 | 4 de trefle |
| 3 | 4 | 15 | 4 de carreaux |
| 0 | 5 | 16 | 5 de pique |
| 1 | 5 | 17 | 5 de cœur |
| 2 | 5 | 18 | 5 de trefle |
| 3 | 5 | 19 | 5 de carreaux |
| etc | etc | etc | etc |

Répartition des cartes 1

La pile « tas » est mélangée en échangeant chaque entier avec un autre entier du tas parmi les 52 à l'aide de la fonction « random() » de la librairie Math ce qui nous permet d'avoir un comportement aléatoire dans la répartition des cartes.

A chaque fois que l'on veut tirer une carte, il suffit de dépiler un élément de « tas » et on n'a donc aucun doublon de cartes possible. Cette action s'effectue par le biais de « tireUneCarte() ».

Lorsque que « deroulement() » commence, il faut initialiser les variables des clients en jeu, les cases du tableau « jetonsPose[] » et les différentes variables permettant le déroulement du jeu. Ce tableau permet de conserver les jetons posés par chaque client même s'il se déconnecte car c'est nécessaire pour répartir les gains correctement.

On peut remarquer qu'il y a plusieurs fonctions permettant d'effectuer les enchères. Même si elles sont assez similaires au premier coup d'œil, elles sont en fait assez différentes. « premiereEnchere() » permet d'effectuer la première enchère qui n'a pas de première phase contrairement aux autres et qui doit faire un tour complet de table y compris du joueur qui a mis la surblind alors que c'est le dernier à avoir relancé. Dans une enchère classique on ne retourne pas sur la dernière personne à avoir relancé.

La « premierePhase() » permet de regarder si quelqu'un relance lorsque l'on vient d'engager un nouveau tour d'enchère et s'il y a une relance alors seulement on lancera « enchere() » sinon on passera au prochain tour d'enchère.

Lors d'un tour d'enchère on va parcourir les joueurs en jeu (attente==0) et envoyer un message aux clients chacun leur tour pour leur demander de faire un choix d'action. Ces choix sont : passer, suivre et relancer (représenté respectivement par l'entier 1, 2 ou 3). La relance n'est pas toujours possible (3 relance max par tour d'enchère, ouvrir n'est pas une relance), le message contient donc un booléen (« true » ou « false ») pour indiquer aux clients s'ils doivent proposer ce choix.

Le serveur va se mettre en attente de la réponse du client pendant 15 secondes en instanciant « TimerChoixJoueur » et en mettant un sémaphore. Si au bout des 15 secondes il n'y a pas de réponse, l'instance lance « jouage() » avec les paramètres pour passer et donc le joueur se couche (attente=1), ça libère le sémaphore et le programme reprend dans choix(). Si le client envoie une réponse correcte, l'instance du client lance « jouage() » avec le choix du joueur, ça libère le sémaphore et le programme reprend dans choix() pour appliquer le choix du joueur changer les paramètres et envoyer les modifications aux clients.

Un mauvais choix implique que le joueur se couche. Un mauvais nombre de jetons lors d'une relance est rétabli : s'il y a plus de jetons que ce qu'il possède alors il fait tapis, s'il y en a moins ou le même nombre que le minimum pour suivre alors son choix devient suivre.

A la fin de tous les tours d'enchère, vient la répartition des gains. C'est là qu'intervient la variable 'pot' des clients. Pour savoir, combien a le droit de toucher chaque joueur, il suffit de faire jetons de la table moins le pot du client.

La fonction « repartitionDesGains() » permet de répartir les gains entre les clients selon leur statut (gagnant, perdant, tapis).

Pour savoir qui sont les gagnants, l'instance Jeu de « PokerPartie » appelle la fonction « gagnant() » qui va comparer les mains de chaque client encore en jeu. Pour ce faire, on va regarder quel est la meilleure combinaison de chaque client en regardant dans cet ordre s'il a :

- Quinte flush royale.
- Quinte flush.
- Carré.
- Full.
- Couleur.
- Suite.
- Brelan.
- Deux paires.
- Une paire.

Pour chaque combinaison, on récupère en même temps la meilleure carte de la combinaison. Puis s'il y a égalité, on va comparer le reste des cartes (je rappelle qu'on ne prend en compte que 5 cartes).

Après la répartition des gains, il faut vérifier si la partie est terminée (plus qu'un seul joueur avec des jetons) et si ce n'est pas le cas relancer un tournoi() continue de boucler et relance le premier tour d'enchère en faisant avancer l'indice du client devant poser la blind.

3) La base de données

Toutes les actions sur la base de données sont gérées via les méthodes de la Classe ClientBDDNoSQL. La classe PokerServer possède un objet de ce type et utilise ses méthodes. MongoDB dispose d'une API java simple et documentée, utilisée ici.

Les méthodes de l'API permettent : de se connecter à la BDD, de faire requêtes de lecture, d'insertions et de modification dans la BDD.

La classe ClientBDDNoSQL possède de nombreuses autres méthodes répondant aux besoins précis envers la BDD qui sont :

- Créer ou effacer un compte pour un joueur.
- Changer le pseudo ou le mot de passe d'un compte.
- Vérifier le mot de passe d'un joueur donné.
- Augmenter le nombre de parties gagnées ou perdues.
- Retourner les infos d'un ou de tous les Joueurs.

Voici le code de la fonction qui permet de se connecter à la BDD, qui est lancé au démarrage du serveur :

```
/**
 * Fonction pour se connecter à la base de données et choisir la collection
 * @author Maurin Benjamin
 */
public void Ouverture(){
    //connection
    Mongo m = null;
    try {
        m = new Mongo( "localhost" , 27017 );
    } catch (Exception ex) {
        ex.printStackTrace();
        Logger.getLogger(ClientBDDNoSQL.class.getName()).log(Level.SEVERE, null, ex);
    }
    //choix de la bdd
    db = m.getDB("pokerBDD");
    //choix de la collection (environ table)
    coll = db.getCollection("pokerCollection");
}
```

Connexion à la base de données

Contrairement aux *API SQL*, ici les requêtes se font en java via de simples fonctions « put » ou « find ». Ceci permet de maîtriser plus facilement les requêtes que l'on veut faire.

Les méthodes de « ClientBDDNoSQL » retournent des String correspondant au résultat des requêtes, en fonction du protocole. « PokerServer » n'a ainsi qu'à envoyer au client ce que retourne les méthodes de « ClientBDDNoSQL ».

Par exemple, lorsqu'un client veut se logger, le *thread* qui réceptionne la demande a ce code :

```
//Demande de connection
if(cmd.equals("CONNECT")){
    screenOut.println("demande de connection\n");
    test1=st.nextToken();
    PokerServer.testJconnect(test1);
    cmd=PokerServer.bd.verifPassword(test1,st.nextToken());
    send(cmd);
    if(!cmd.equals("CONNECTOK")){
        lecture=false;
    }
    else {connecte = true;pseudo = test1;}
    return 1;
}
```

Thread qui gère la connexion à la base de données

Dans ce code, l'objet bd de « PokerServer » est une instance de « ClientBDDNoSQL » précédemment connecté au SGBD NoSQL.

Voici la méthode « `verifPassword` » de cet objet :

```
/**
 * Fonction permettant de vérifier les informations de connexion du joueur.<br><br>
 *
 * Retourne :<br>
 * <ul>
 * <li> <b>CONNECTOK</b> : opération effectuée </li>
 * <li> <b>WPASS</b> : mauvais password utilisé, connexion impossible </li>
 * <li> <b>WPSEUDO</b> : pseudo introuvable, connexion impossible </li>
 * </ul>
 *
 * @author Maurin Benjamin
 *
 * @param pseudo pseudo du joueur voulant se connecter
 * @param motDePasse mot de passe du joueur voulant se connecter
 *
 * @return {@code String} contenant le résultat de l'opération
 */
public String verifPassword(String pseudo, String motDePasse){
    DBCursor cur;
    DBObject courant;
    BasicDBObject query = new BasicDBObject();
    query.put("pseudo", pseudo);
    cur = coll.find(query);

    while(cur.hasNext()) {
        courant = cur.next();
        if(courant.get("password").equals(motDePasse)) {
            return "CONNECTOK";
        }else{
            return "WPASS";
        }
    }
    return "WPSEUDO";
}
```

Méthode de vérification

Ici, une requête est créée associant le message pseudo avec la variable pseudo passée en paramètre. Le `DBCursor` `cur` permet de parcourir les résultats de cette requête (via la fonction `.next()`). Chaque résultat est de la forme `DBObject` qui est l'équivalent d'un tuple en SQL. Il suffit alors de comparer l'attribut « `password` » avec celui fourni par le client.

4) Le lecteur de commandes

Le lecteur de commande doit être actif à tout moment. Pour cela, La classe « `CommandReader` » (qui est un *thread*) est lancée au lancement du serveur, et elle boucle sur l'écoute des commandes de l'utilisateur.

Comme pour le traitement des messages, cette classe découpe la commande pour effectuer la bonne action et utilise les fonctions statique de « `PokerServer` » pour les mener à bien.

Voici un morceau de la fonction qui traite les commandes :

```
/**
 * Fonction qui lit et traite les commandes
 * @author Benjamin Maurin
 */
@Override
public void run(){
    boolean valid;
    try{
        while ((command = in.readLine()) != null){
            valid = false;
            command = command.trim().toUpperCase();
            if (command.compareTo("QUIT")==0){
                valid = true;
                PokerClientThread foo;
                for (int i=0;i<PokerServer.clientList.size();++i){
                    foo= (PokerClientThread) PokerServer.clientList.get(i);
                    foo.deco();
                }
                PokerServer.serverSocket.close();
                System.exit(0);
            }
            if (command.compareTo("LIST")==0){
                valid = true;
                screenOut.println("");
                PokerServer.listClients();
                screenOut.println("");
            }
        }
    }
}
```

Fonction de traitement des commandes

5) Diagrammes de l'application

- Diagramme des cas d'utilisation :

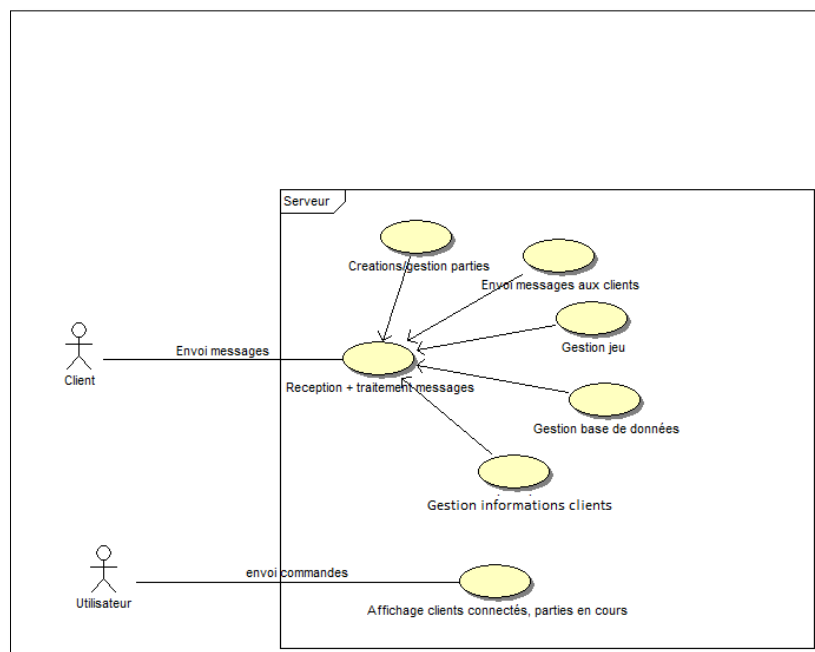


Diagramme des cas d'utilisations

- Diagramme des classes :

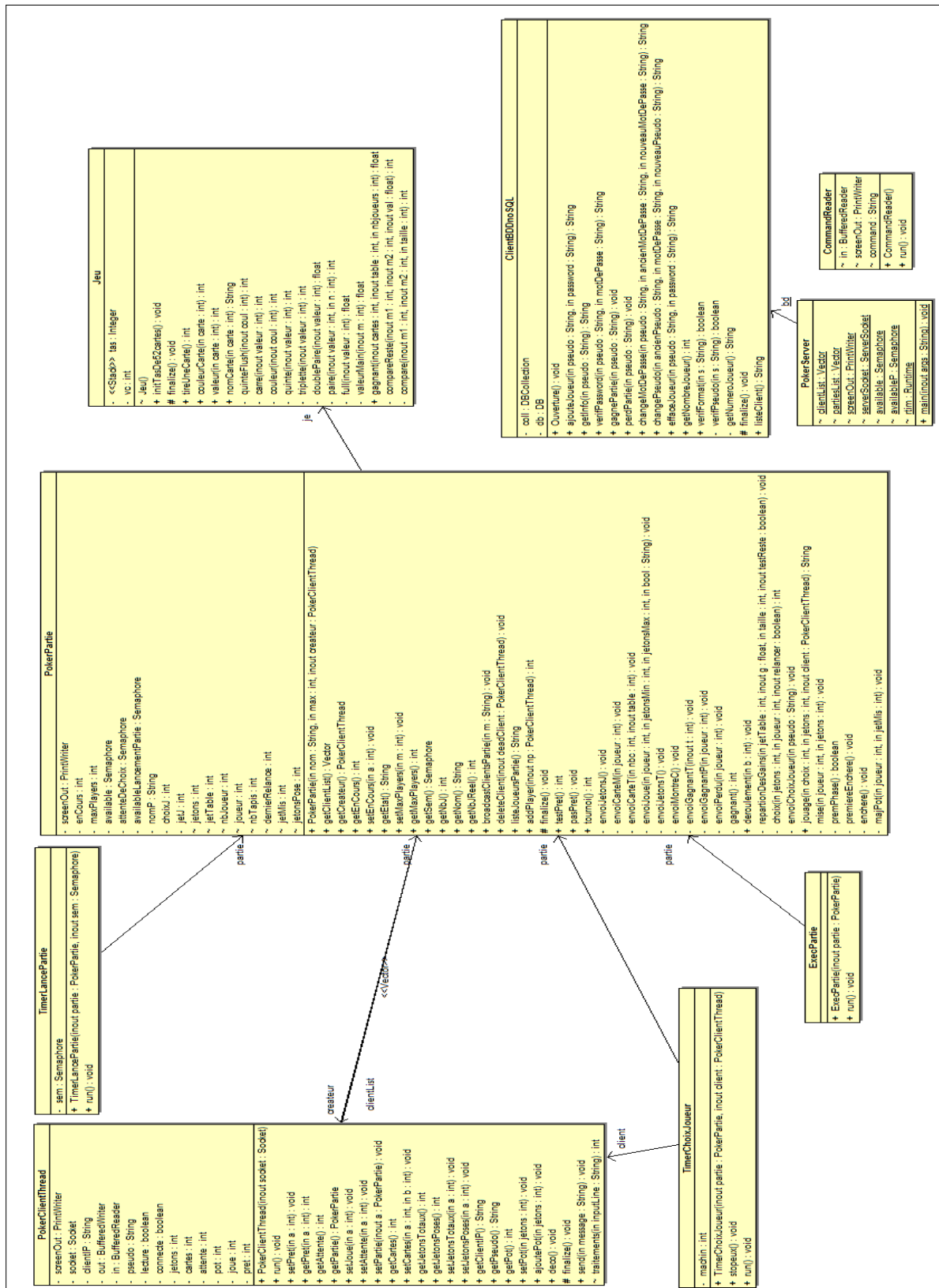


Diagramme des classes

F) Fonctionnement global du projet

Cette partie va décrire le fonctionnement général du projet, c'est à dire de quelle manière l'ensemble des parties (des clients et du serveur) interagissent entre elles pour faire fonctionner le jeu.

1) Communication Client-Serveur

Le but du projet est que le jeu de poker soit multi plate-forme. Le fonctionnement de chacun des clients doit donc être standardisé pour que le serveur fonctionne pour n'importe quel client et que si nécessaire, on puisse par la suite rajouter de nouveaux clients.

Pour que chaque client puisse communiquer avec le serveur, il faut que le type de message et les messages en eux-mêmes soient standardisés,

Les clients et le serveur communiquent donc avec des *Socket*, type d'objet disponible dans de nombreux langages. Chaque client et le serveur peuvent ainsi recevoir et comprendre les messages envoyés.

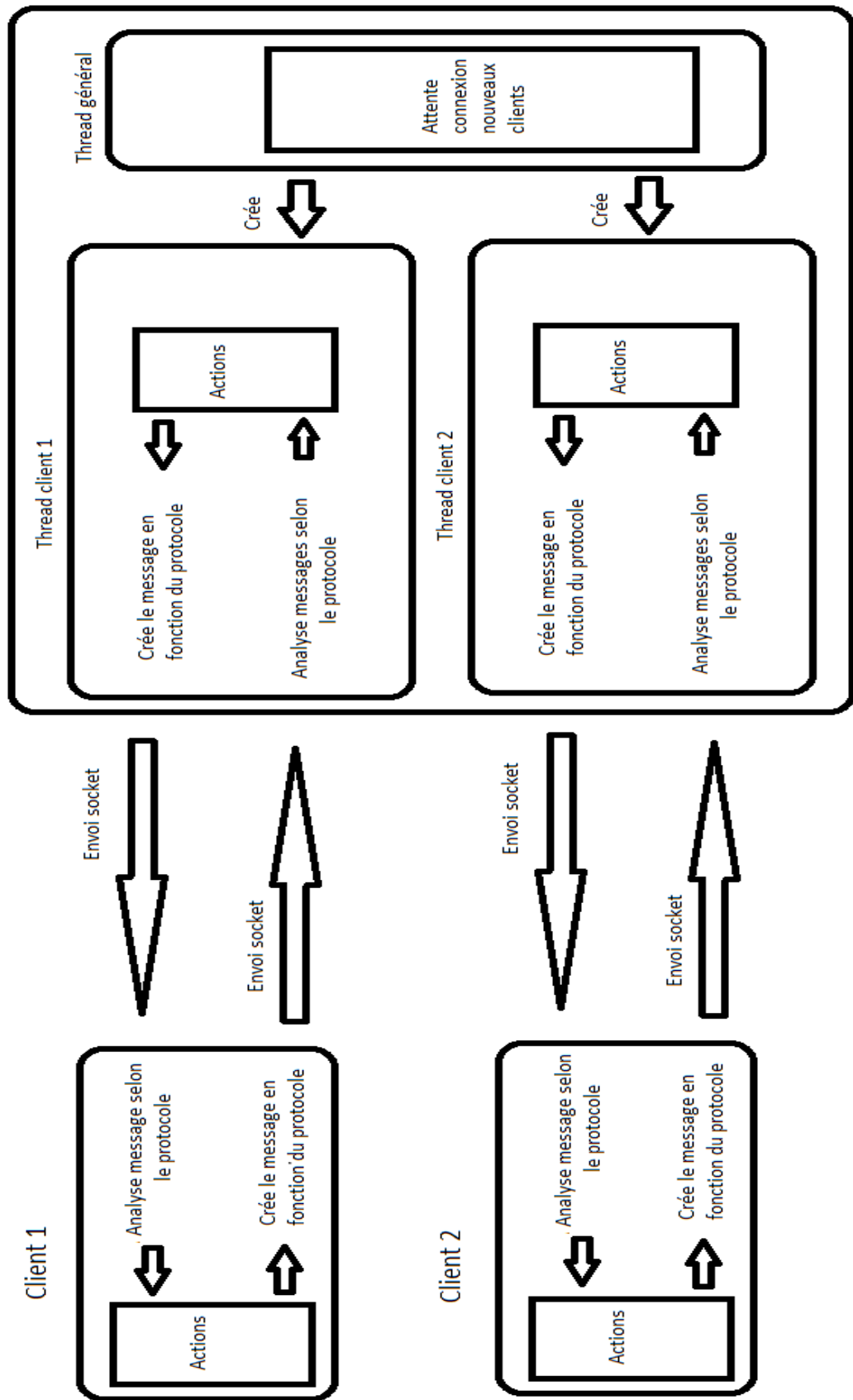
Pour que les messages de tous les clients puissent être traités de la même manière, nous utilisons un protocole strict (protocole en annexe), et les messages qui ne correspondent pas au protocole sont rejetés.

La communication se déroule en mode connecté, c'est-à-dire qu'un client doit se connecter au serveur, puis pourra communiquer avec lui.

De plus, le client ou le serveur peut fermer le circuit virtuel entre les deux pour diverses raisons comme perte de liaison internet, si un client est trop long à répondre présent lors d'un démarrage de partie etc...

Pour que le serveur puisse communiquer avec plusieurs clients, lorsqu'il reçoit la demande de connexion d'un nouveau client, il instancie une classe qui contiendra les informations du client connecté, et un *thread* pour pouvoir communiquer avec un client et continuer ses autres traitements.

Serveur



Communication Client/serveur

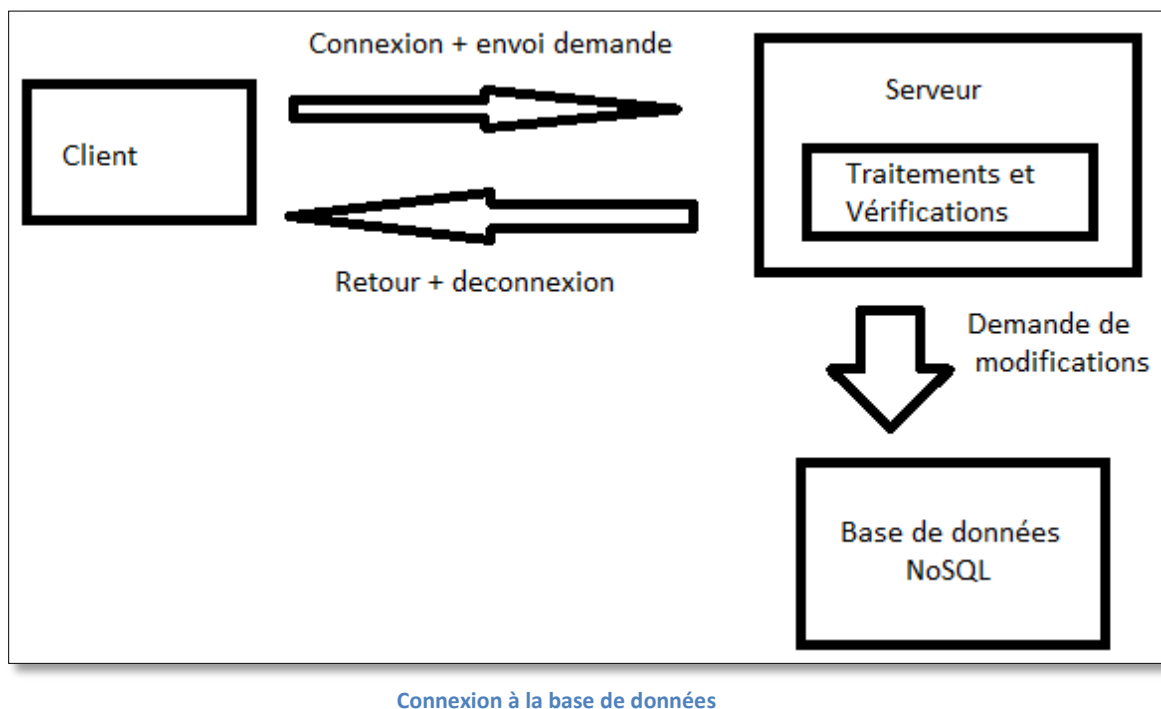
2) Déroulement d'une partie

Il y a trois types d'actions que peuvent effectuer les clients : les actions pré-parties, les actions pour le commencement d'une partie, et les actions du déroulement de la partie.

- Les actions pré-parties :

Chaque client peut, avant une partie, créer un compte, changer le mot de passe ou le pseudo du compte. Pour cela, le client se connecte, envoie les informations nécessaires puis est immédiatement déconnecté.

Pour qu'un compte soit créé, le pseudo et le mot de passe correspondant doivent respecter un format (de 2 à 15 caractères sans caractères spéciaux), et qu'un compte avec le même pseudo n'existe pas. La date de création du compte est conservée dans la base de données.



Les clients peuvent aussi se connecter physiquement au serveur, puis se connecter à leur compte en envoyant leur pseudo et leur mot de passe. Une fois « loggés » ils pourront effectuer les actions pour le commencement d'une partie.

Si un client se connecte alors qu'un client ayant le même compte est déjà loggé, alors ce dernier est déconnecté, permettant ainsi d'éviter les clients connectés et inactifs ainsi que d'avoir des clients au même pseudo.

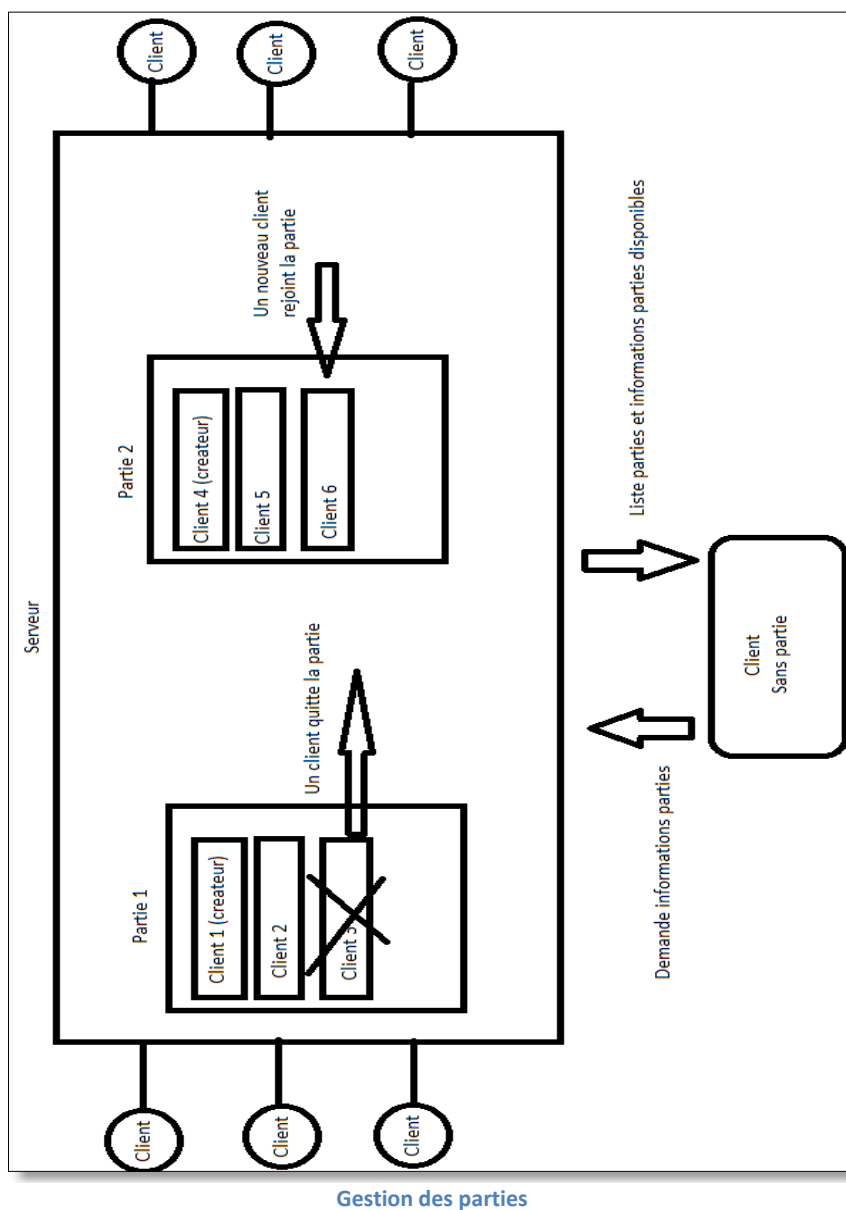
- Les actions pour le commencement:

Une fois « loggés », les clients peuvent :

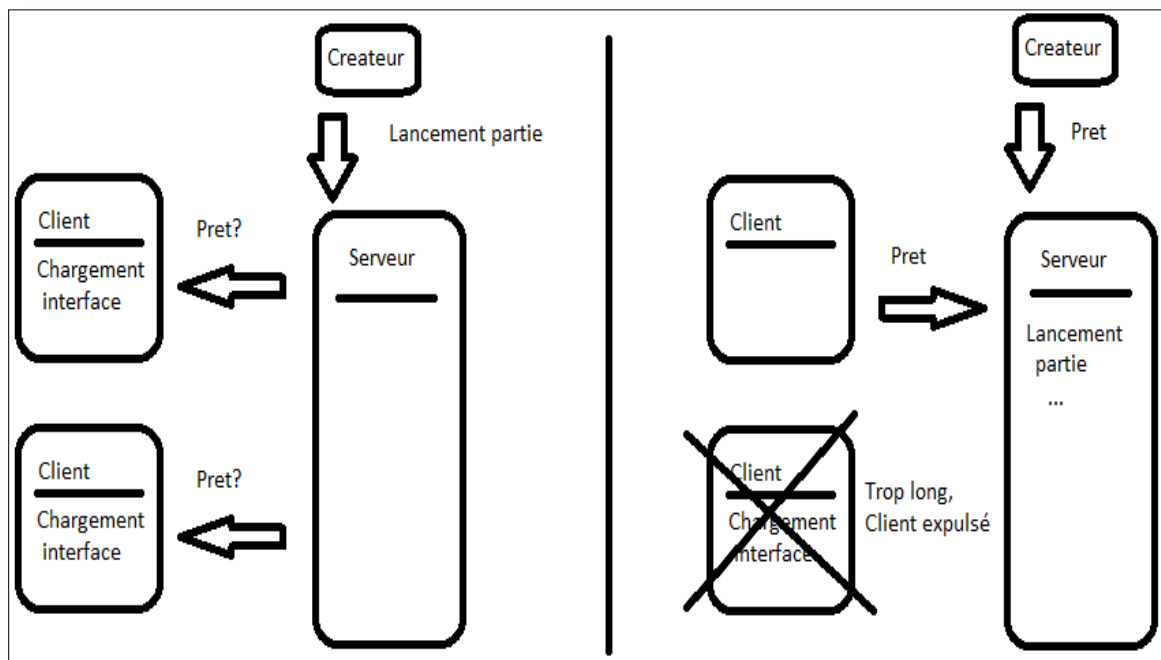
- Demander des informations sur les parties en attentes (nom, nombre de joueurs).
- Demander des informations sur un autre client connecté.
- Créer une partie.
- Rejoindre une partie.

Une fois dans la partie, les clients reçoivent la liste des personnes présentes dans leur partie et attendent que le créateur de la partie la démarre.

Si le créateur de la partie quitte la partie, un autre client sera désigné comme créateur. S'il n'y a plus personne dans la partie, elle est détruite.



Lorsque le créateur de la partie demande au serveur de lancer la partie (qui doit contenir au moins 2 clients), le serveur prévient tous les clients de la partie que la partie va commencer. Les clients chargent leur interface de jeu puis envoient un message au serveur pour prévenir qu'ils sont prêts. Les clients qui mettent plus de cinq secondes à répondre sont expulsés de la partie.



Création d'une partie

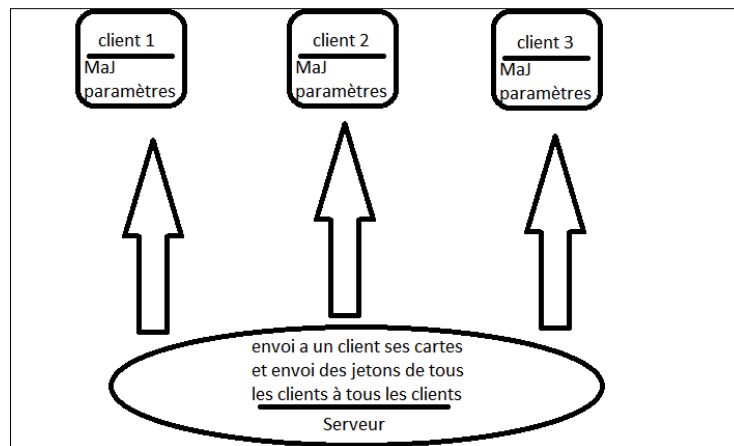
- Les actions du déroulement de la partie

La partie suit les règles du *Texas hold'em no limit* en mode tournoi. Pour résumer, le but du jeu est que tous les autres joueurs soient éliminés de la partie en perdant tous leurs jetons. Il y a 2 cartes distribuées à chaque joueur et 5 seront distribuées sur la table. La meilleure combinaison de 5 cartes gagne la manche. Il n'y a pas de limite de mise mais on ne peut relancer que 3 fois par tour d'enchère.

La partie commence par l'initialisation des jetons de chaque joueur à 200 et en mettant toutes les autres caractéristiques à 0 (pot, jetons posé sur la table, état (en jeu, tapis, perdu, couché)). Ensuite le serveur désigne qui va poser la *blind* et qui va poser la *surblind* (au début les 2 premiers de la liste) puis distribue les 2 cartes de la main de chaque joueur.

Chaque action concernant les joueurs fait l'objet d'un message envoyé aux clients pour qu'ils mettent à jour le leur de leur côté.

Le serveur envoie donc ici un message à chaque client avec ses cartes (celle que le client doit afficher dans la main du joueur) et jetons de chaque client à chaque clients, ici 200 pour que chaque client puisse afficher les jetons de tous les joueurs.

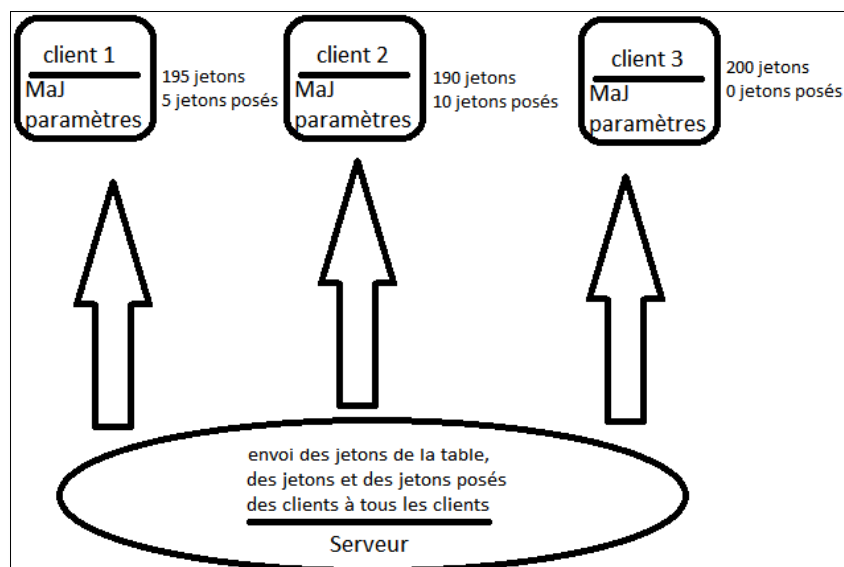


Communication avec les clients

Nous pouvons maintenant commencer le premier tour de jeu (on va définir le tour de jeu comme commençant après que les deux cartes de la main soient distribuées et se terminant après la comparaison des mains (s'il y en a besoin) et la répartition des jetons au(x) gagnant(s) du tour).

La blind et la surblind sont posées par les joueurs désignés (5 et 10 jetons). Le serveur retire donc 5 et 10 jetons à ces joueurs et ajoute ces mêmes valeurs au nombre de jetons qui sont sur la table. On a donc maintenant 15 jetons sur la table et les deux premiers joueurs ont 195 et 190 jetons et 5 et 10 jetons posés.

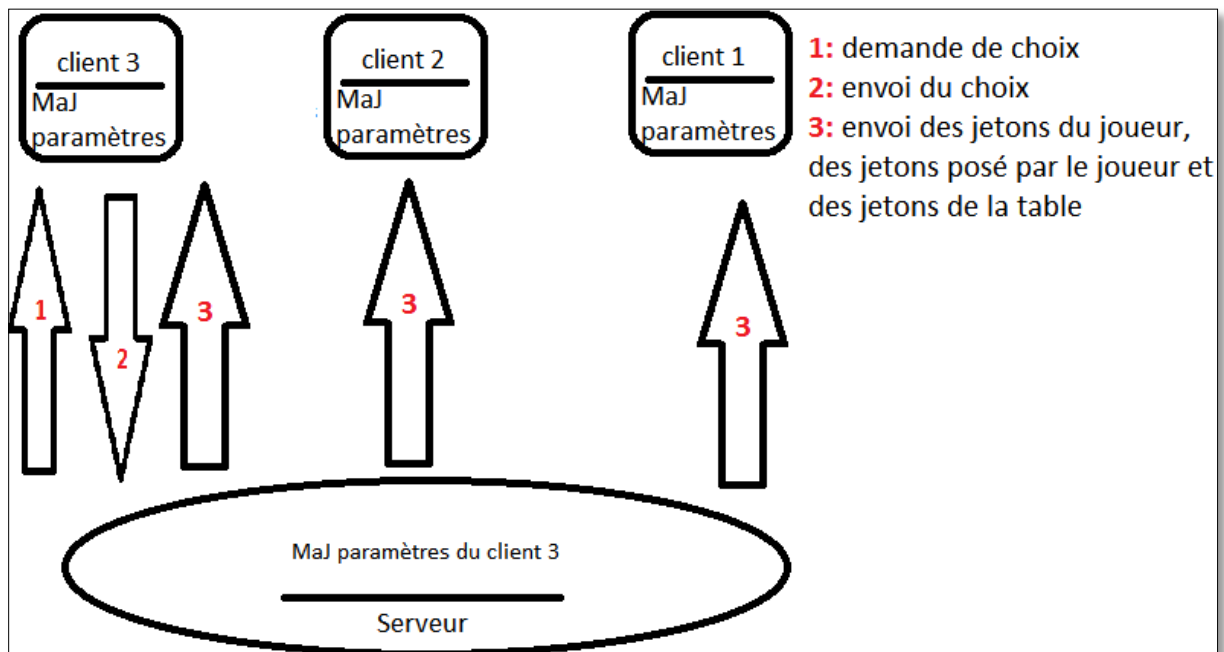
Toutes ces informations sont donc envoyées à chaque client pour leur permettre de mettre à jour leurs données.



Actualisation des données de joueur

Ceci s'enchaîne avec le premier tour d'enchère. On fait le tour de chaque joueur en commençant par le joueur qui suit la surblind (ici le client 3). On lui demande ce qu'il veut faire en lui envoyant un message avec ses choix possible et les jetons minimum et maximum qu'il peut miser. Le client a le choix entre se coucher, suivre/check ou relancer.

Le serveur se met en attente du choix du joueur pendant 15 secondes. Si au bout de cette attente le client n'a pas renvoyé de message de choix, le serveur le fait se coucher sinon il applique le choix du joueur. Ensuite le serveur envoie un message pour indiquer les modifications des paramètres du joueur et on passe au joueur suivant et on lui demande son choix et ainsi de suite.



Principes de communication

A la fin du tour d'enchère, 3 cartes sont tirées pour la table et envoyées aux clients et on refait un tour d'enchère. Puis on continue en tirant une carte, un tour d'enchère, la dernière carte et le dernier tour d'enchère. Bien sûr, les tours d'enchères ne se font que si nécessaire (encore au moins 2 joueurs peuvent miser (c'est-à-dire qu'ils ne sont ni couché, ni en tapis, ni défait)).

S'il ne reste qu'un seul joueur qui n'est pas couché, alors il gagne les jetons de la table. Sinon, il faut voir qui est le gagnant du tour de jeu. Le serveur alors calcule qui est le gagnant ou qui sont les gagnants en cas d'égalité puis met à jour les données des clients et envoie un message à un client pour leur indiquer quelles sont les cartes des autres joueurs, qui est le gagnant et les changements effectués sur les jetons de chacun. En cas de tapis, si le client gagne, il ne recevra que ce qu'il a le droit de toucher (jetons misés par les clients inférieur ou égal à son tapis) et le reste des jetons est réparti sur les autres gagnants ou les joueurs qui ne sont pas couchés.

A la fin de chaque tour de jeu, on vérifie si des clients se retrouvent avec zéro jeton. Si c'est le cas alors ces joueurs ont perdu et ne peuvent plus jouer, mais ils peuvent rester en tant que spectateur. A chaque fois qu'il y a un nouveau perdant le serveur envoie un message à tous les clients pour leur indiquer.

S'il ne reste plus qu'un seul joueur en jeu alors la partie se termine et ce joueur gagne la partie. On envoie donc un message à tous les clients pour leur indiquer le gagnant et la partie est détruite. Par contre, s'il reste plusieurs joueurs en jeu, un nouveau tour de jeu se lance et on recommence en assignant la mise de la blind au joueur suivant l'ancienne blind.

V. Structure organisationnelle

A) Planning

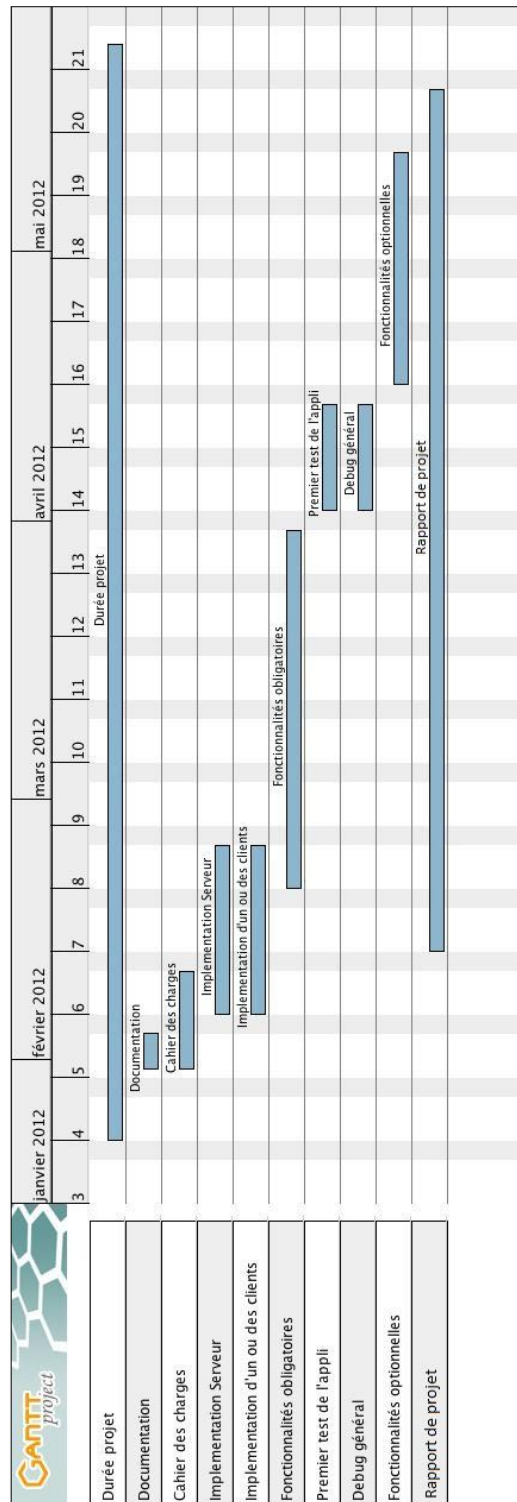


Diagramme de Gantt de notre projet

B) Organisation et fonctionnement du groupe de travail

1) Fonctionnement avec le tuteur

Nous avons eu cinq réunions avec notre tuteur M. MEYNARD pour savoir si nous répondions à ses attentes et si les décisions que nous prenions étaient bonnes ou non (choix du SGBD ou du type de jeu par exemple).

Nous avons aussi pu fréquemment le contacter par e-mail lorsque l'on avait des petites questions concernant le projet ou pour voir si un rendez-vous était possible si nous en avions besoin.

2) Fonctionnement au sein du groupe

Notre groupe étant composé de huit personnes, nous avons pu répartir le travail afin d'être plus productif. Notre projet pouvait se diviser en 4 parties bien distinctes :

- Le serveur,
- Le client Android,
- Le client JavaScript,
- Le client C++.

Nous avons décidé, lors de notre première réunion, d'assigner 2 personnes à chaque tâche. Pour ce faire, nous avons commencé par effectuer un premier tour de table pour voir si certains avaient des préférences, puis nous avons discuté pour que tout le monde choisisse sa partie.

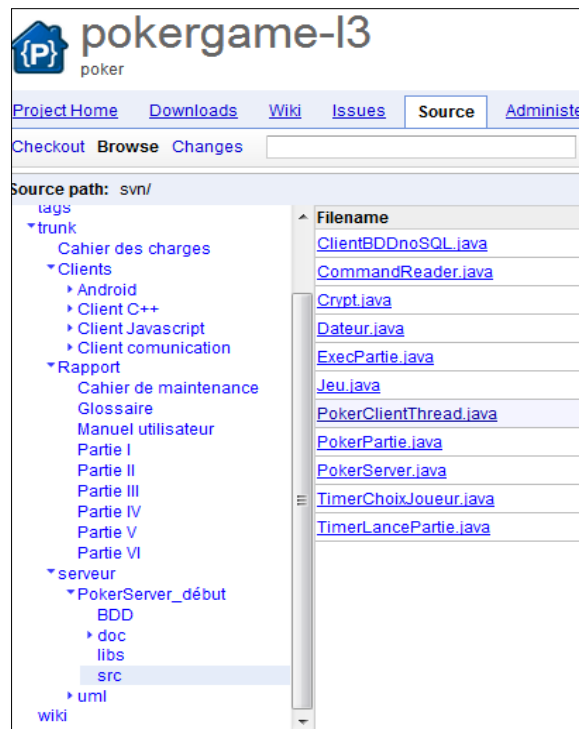
Et finalement, nous avons trouvé un commun accord avec cette composition :

- Benjamin MAURIN et Steve GINER sur le serveur,
- Jessy BONNOTE et Mathieu POLIZZI sur le client Android,
- Paul MURA et Yohann LAMSECK sur le client web,
- Clément AGRET et Renaud LE GOC sur le client C++.

Lors de cette première réunion, nous nous sommes aussi répartis les parties concernant le cahier des charges. Ceci nous a permis, une semaine plus tard, de faire une réunion sur les différents travaux à accomplir grâce à une étude de l'existant. Lors de cette réunion nous avons décidé de quelles seraient les fonctionnalités obligatoires et celles optionnelles afin d'avoir une idée précise des objectifs à tenir.

Ensuite, la répartition des tâches s'est faite au sein de chaque binôme puis nous nous réunissions régulièrement (une semaine sur deux en général) pour faire le point sur l'avancement de chaque groupe, pour se concerter sur les protocoles de communication client-serveur, répondre aux différentes questions qu'un membre pouvait avoir et tester les nouvelles fonctionnalités.

Pour partager les données, et voir comment tous les groupes avançaient, nous avons utilisé un SVN (ToirtoiseSVN). Ceci nous a permis de gérer les versions et de pouvoir travailler chacun de son côté sans difficulté. Cet outil a donc nettement augmenté notre vitesse de travail.



Aperçu du SVN

Voici l'arborescence de nos données sur le SVN et un exemple des logs montrant des « commits » (dépôts) effectués sur le serveur. Pour récupérer les dernières données déposées par les membres, soit on allait sur le site, soit on effectuait simplement un « update » (mettre à jours) avec « ToirtoiseSVN ».

| Project Home Downloads Wiki Issues Source Administer | | | | |
|--|--------|---|--------------|-------------------------|
| Checkout Browse Changes <input type="text"/> <input type="button" value="Search Trunk"/> Request code review | | | | |
| Committed Changes Newer 99 - 75 of 199 Older | | | | |
| Rev | Scores | Commit log message | Date | Author |
| r99 | | [No log message] | Mar 12, 2012 | paul.mura.paroche |
| r98 | | Plein de changements serveur dont parties qui se delr | Mar 10, 2012 | Ben.Maurin@gmail.com |
| r97 | | [No log message] | Mar 8, 2012 | paul.mura.paroche@... |
| r96 | | [No log message] | Mar 8, 2012 | paul.mura.paroche@... |
| r95 | | avancement de class PokerPartie pour gerer le deroul | Mar 7, 2012 | giner.steve@gmail.com |
| r94 | | Legers changements serveur | Mar 6, 2012 | Ben.Maurin@gmail.com |
| r93 | | Client QT | Mar 2, 2012 | clement.agret@gmail.... |
| r92 | | Rajout commande serveur pour voir les credits | Feb 29, 2012 | Ben.Maurin@gmail.com |
| r91 | | Rajout fonctionnalité commande KILL pour déconnecter | Feb 28, 2012 | Ben.Maurin@gmail.com |
| r90 | | Ya eu un bug dans le commit des docx | Feb 28, 2012 | Ben.Maurin@gmail.com |
| r89 | | Rajout dossier Rapport (et sous-fichiers que chacun p | Feb 28, 2012 | Ben.Maurin@gmail.com |
| r88 | | ya plus d'accents ca va mieux ? | Feb 26, 2012 | giner.steve@gmail.com |
| r87 | | Changements serveur ajout fonctions pour créer et rej | Feb 25, 2012 | Ben.Maurin@gmail.com |
| r86 | | Changements serveur+ STEVE ENLEVE LES é DE LA | Feb 25, 2012 | Ben.Maurin@gmail.com |
| r85 | | avancement des fonctions sur les cartes et test des fo | Feb 25, 2012 | giner.steve@gmail.com |
| r84 | | Projet poker V1.03 Application compatible avec tout les | Feb 23, 2012 | shyzkanza@gmail.com |
| r83 | | Plein de changements serveur | Feb 22, 2012 | Ben.Maurin@gmail.com |
| r82 | | Mise en ordre uml serveur | Feb 20, 2012 | Ben.Maurin@gmail.com |
| r81 | | Modification uml et rajouts diagrammes cas d'utilisatic | Feb 18, 2012 | Ben.Maurin@gmail.com |
| r80 | | Ajout semaphores dans clientBDD + ajouts variables r | Feb 18, 2012 | Ben.Maurin@gmail.com |
| r79 | | corection faute, mise en page cahier des charge, l'uml | Feb 18, 2012 | giner.steve@gmail.com |
| r78 | | Projet poker V1.02 Application compatible avec tout les | Feb 17, 2012 | shyzkanza@gmail.com |
| r77 | | [No log message] | Feb 17, 2012 | shyzkanza@gmail.com |
| r76 | | avancement des classes pour determiner les valeurs | Feb 16, 2012 | giner.steve@gmail.com |
| r75 | | Suppressions fichiers cryptage + modification serveur | Feb 15, 2012 | Ben.Maurin@gmail.com |
| Newer 99 - 75 of 199 Older | | | | |

LOGS du SVN

Nous avons beaucoup travaillé avec Skype, un logiciel de communication par Internet. Cet outil permet de faire des visio-conférences et du partage d'écran, ce qui nous a permis de travailler en groupe même depuis chez nous et de tester nos codes en situation réelle (en ligne). Il suffisait que l'un de nous lance le serveur, fournisse son IP et les clients pouvait se connecter au serveur. Nous avons donc pu débbugger au fur et à mesure nos codes et avancer plus rapidement.

VI. Conclusion

A) Bilan

Notre projet avait pour but la création d'un jeu de poker avec 3 clients différents (C++, Web et Android) et un serveur. Au final, nous avons créé deux clients fonctionnels avec leur interface (Android et Web), un client sans interface mais fonctionnel (C++) et un serveur (en Java) fonctionnel permettant de gérer plusieurs parties à la fois. Nous avons donc créé un jeu de poker en ligne permettant de faire des tournois de Texas hold'em no limit.

Nous nous étions fixés des fonctions principales et secondaires pour nous permettre d'avoir un programme fonctionnel a présenté et pouvant être récupéré puis amélioré par d'autres développeurs. Nous nous sommes donc concentrés sur les fonctions principales indispensables avec comme objectif un code clair et commenté (JavaDocs disponibles) pour simplifier le futur travail de ce qui le reprendrons.

Nous n'avons pas réussi à faire tout ce que nous avions prévu à cause de nombreuses difficultés rencontrées au cours de la programmation du logiciel, surtout en C++, et à la coordination du groupes. En effet, c'était la première fois que nous étions confrontés à un projet en assez gros groupe (8 personnes).

B) L'apport de ce projet

La réalisation de ce projet nous a beaucoup apporté et a approfondis nos compétences dans différents domaines. Ce projet nous a appris à :

- Répartir les tâches entre différentes personnes et les difficultés que ceci peut entraîner.
- Nous servir d'un SVN pour partager/sécuriser les données et voir l'avancement global du projet.
- Faire en sorte que le code soit récupérable.
- Maîtriser de nouveaux langages de programmation où approfondir nos compétences dans un langage connu.

Enfin, cela nous a obligé à rédiger un rapport conséquent et structuré. Tout ceci aura donc été très bénéfique pour des projets futurs que ce soit dans les études ou dans la vie professionnelle.

C) Améliorations possibles du logiciel

Le logiciel côté serveur comme client peut subir de nombreuses améliorations telles qu'un ajout de calcul de probabilité de gagner en fonction des cartes visibles. On pourrait, par exemple, aussi ajouter de nouvelles règles de poker pour permettre aux joueurs de choisir quel type de partie il veut faire.