Projet de Réseaux - Master 1 Informatique

Cyrille d'Halluin - Arthur Marzinkowski

Université d'Artois - Faculté des Sciences Jean Perrin

Introduction

Ce projet a pour but de créer un serveur de quiz proposant à ses utilisateurs de participer à des sessions de questions-réponses multijoueurs. Le projet consiste à développer d'une part le serveur de quiz lui-même et d'autre part le client permettant à des utilisateurs de se connecter au serveur afin de jouer les uns contre les autres.

Informations générales

- Le projet se fera en binôme ou en trinôme au choix des étudiants.
- La partie serveur devra être écrite en C (ou en C++). La partie cliente sera écrite dans un autre langage au choix du binôme/trinôme
- Il n'est absolument pas obligatoire d'intégrer une interface graphique sophistiquée dans la partie cliente. Il est tout à fait possible de développer l'application avec une interface simple en mode texte.
- Le projet sera mis à disposition sous Git. N'oubliez pas d'ajouter vos enseignants de cours et TP en tant que reporter.

Vous présenterez votre projet dans la semaine du **15 au 19 décembre 2025**. Cette présentation d'une quinzaine de minutes sera faite en binôme/trinôme et suivant un planning qui vous sera communiqué ultérieurement. Il conviendra d'installer quelques minutes avant votre passage le client et le serveur sur l'une des machines de la salle P309. Client et serveur devront évidemment s'exécuter sans problème sur ces machines. A l'issue de cette présentation, votre projet **disponible et accessible sous git** sera considéré comme rendu.

Qu'attendons-nous de vous ?

Nous serons particulièrement attentifs aux points suivants :

- Qualité du code (commentaires, organisation, etc...)
- Choix des protocoles réseaux utilisés en fonction des besoins
- Qualité du protocole de communication (robustesse, fiabilité,...) Par exemple : que se passe-t-il si le serveur tombe ? Que se passe-t-il si un joueur se déconnecte en pleine partie ?
- Facilité d'utilisation, compréhension du système pour un utilisateur lambda
- Gestion correcte de la synchronisation entre joueurs

Important - Politique d'intégrité académique

Utilisation des intelligences artificielles génératives

L'utilisation d'outils d'intelligence artificielle générative (ChatGPT, Copilot, etc.) pour générer automatiquement le code de ce projet est **strictement interdite** et sera considérée comme de la fraude académique. Tout manquement à cette règle entraînera l'annulation du projet.

Vous êtes toutefois autorisés à :

- Consulter la documentation officielle des langages et bibliothèques
- Rechercher des solutions à des problèmes spécifiques sur StackOverflow ou forums similaires
- Utiliser les IDE avec autocomplétion standard

Présentation du jeu QuizNet

Dans "QuizNet", les joueurs participent à des sessions de quiz en temps réel sur différents thèmes. Chaque session peut accueillir plusieurs joueurs qui répondent simultanément aux mêmes questions. Le système gère deux modes de jeu permettant de varier l'expérience :

- **Mode Solo** : chaque joueur joue pour lui-même, le meilleur score gagne
- **Mode Battle** : chaque joueur dispose de 4 vies. Une vie est perdue lorsqu'un joueur répond incorrectement ou lorsqu'il répond en dernier à une question. Le dernier joueur ayant encore des vies remporte la partie. Le score peut néanmoins être calculé à des fins statistiques mais n'est pas utilisé pour déterminer le vainqueur.

Il n'y a, a priori, rien d'imposé sur le nombre limite de joueurs pouvant participer ensemble à une même session. Vous devez cependant être capable de gérer au minimum 4 joueurs simultanés.

Les thèmes de quiz

Un thème est simplement caractérisé par :

- Un **identifiant unique**
- Un **nom** (ex : "Culture Générale", "Informatique", "Histoire", etc.)

Vous êtes libres de créer autant de thèmes différents que vous voulez. Même si vous décidez de ne créer que deux thèmes, votre client et votre serveur devront être capables d'en gérer plusieurs.

Les questions

Chaque question stockée dans le système est caractérisée par les éléments suivants :

- Un **identifiant unique**
- Un ou plusieurs **thèmes** auxquels elle appartient (une question peut appartenir à plusieurs thèmes)
- Un **niveau de difficulté** (facile, moyen, difficile)
- Un **type** de question (QCM, Vrai/Faux, Texte libre)
- L'**énoncé** de la question
- La ou les **réponse(s) correcte(s)**

Vous devez créer au minimum 30 questions au total, réparties sur au moins 2 thèmes différents avec un équilibre entre les différents niveaux de difficulté.

Types de questions

Il existe trois types de questions différentes :

- **QCM (Questions à Choix Multiples)** : une question avec 4 propositions de réponses dont une seule est correcte. Le joueur doit sélectionner la bonne réponse parmi les 4 propositions.
- **Vrai/Faux** : une affirmation pour laquelle le joueur doit indiquer si elle est vraie ou fausse.
- **Texte libre** : une question pour laquelle le joueur doit saisir une réponse textuelle courte. La vérification se fera par correspondance exacte (insensible à la casse) ou par liste de réponses acceptées.

Définition d'un quiz

Un quiz est défini au moment de la création d'une session par les paramètres suivants :

- Un ou plusieurs **thèmes** : les questions seront piochées parmi celles appartenant à ces thèmes
- Un **niveau de difficulté** (facile, moyen, difficile) : seules les questions de ce niveau seront sélectionnées
- Un **nombre de questions** (entre 10 et 50)
- Un **temps de réponse maximal** (en secondes) qui s'appliquera à toutes les questions de ce quiz

Le serveur sélectionnera aléatoirement les questions correspondant à ces critères au démarrage de la session.

Système de scoring

Le système de scoring fonctionne comme suit :

Points de base

Le nombre de points attribués pour une bonne réponse dépend uniquement du niveau de difficulté de la question :

- **Question facile** : 5 points
- **Question moyenne** : 10 points
- **Question difficile** : 15 points

En cas de mauvaise réponse ou d'absence de réponse (timeout), le joueur ne gagne aucun point.

Bonus de rapidité

Un bonus de rapidité est appliqué si le joueur répond dans la première moitié du temps imparti :

- **Question facile** : +1 point
- **Question moyenne** : +3 points
- **Question difficile** : +6 points

Par exemple, si une question difficile doit être répondue en 30 secondes maximum, un joueur qui répond correctement en moins de 15 secondes gagnera 15 + 6 = 21 points.

Note sur le mode Battle

En mode Battle, le score est calculé selon les mêmes règles mais n'est pas utilisé pour déterminer le vainqueur. Seul le système de vies compte pour déterminer le gagnant. Le score peut néanmoins être affiché et enregistré.

Jokers

Chaque joueur dispose de jokers utilisables une seule fois par session :

- **50/50** (disponible x1) : pour une question QCM, élimine 2 mauvaises réponses au hasard
- **Passer** (disponible x1) : permet de sauter une question sans pénalité (pas de perte de vie en mode Battle)

Les jokers ne peuvent être utilisés qu'au début d'une question (avant que le joueur ne soumette sa réponse).

Déroulement d'une session

Une session de quiz se déroule selon les étapes suivantes :

1. **Création ou rejoindre une session** : un joueur crée une nouvelle session ou rejoint une session existante en attente de joueurs
2. **Salle d'attente** : les joueurs se retrouvent dans une salle d'attente. Le créateur de la session peut :
   - Choisir le ou les thèmes du quiz
   - Définir le niveau de difficulté
   - Définir le nombre de questions (entre 10 et 50)
   - Définir le temps de réponse maximal (entre 10 et 60 secondes)
   - Choisir le mode de jeu (Solo ou Battle)
   - Lancer la session quand il y a au moins 2 joueurs
3. **Déroulement des questions** : les questions sont posées une par une à tous les joueurs simultanément. Pour chaque question :
   - La question est affichée avec son timer
   - Les joueurs répondent dans le temps imparti
   - À la fin du temps ou quand tous les joueurs ont répondu, les bonnes réponses sont révélées
   - Le classement temporaire est affiché
   - En mode Battle, le joueur ayant répondu en dernier ou ayant donné une mauvaise réponse perd une vie. Si un joueur n'a plus de vie, il est éliminé.
   - Pause de 5 secondes avant la question suivante
4. **Fin de session** :
   - En mode Solo : après la dernière question, le classement final basé sur les scores est affiché
   - En mode Battle : la partie se termine quand il ne reste qu'un seul joueur avec des vies, ou après la dernière question (auquel cas le joueur avec le plus de vies restantes gagne)

   Les joueurs retournent ensuite au menu principal.

Lancement du système

Votre projet s'appuie sur une architecture client/serveur. Chaque client se connecte à un serveur QuizNet et échange avec lui suivant un protocole bien défini. Chaque serveur peut accueillir plusieurs clients et gérer plusieurs sessions simultanément. Il est tout à fait possible que plusieurs serveurs tournent simultanément : chaque client a alors le choix de se connecter au serveur qu'il désire.

Lors de son lancement, le client va dans un premier temps rechercher les serveurs QuizNet disponibles (via UDP broadcast). L'utilisateur choisira alors le serveur sur lequel il veut jouer.

Une fois connecté à un serveur, le client peut :

- S'inscrire (créer un compte avec pseudo et mot de passe)
- Se connecter (avec ses identifiants)
- Consulter la liste des sessions disponibles
- Créer une nouvelle session
- Rejoindre une session existante

Données échangées entre client et serveur / Protocole

Remarque générale

Dans cette partie, pour des raisons de clarté, le JSON est formatté correctement avec des retours à la ligne. Ces retours à la ligne ne sont pas nécessaires lors de vos communications. Pour le traitement du JSON, vous pouvez, si vous le souhaitez, utiliser une librairie en C.

Nous listons ci-dessous les données échangées entre client et serveur en fonction des différents besoins. Nous attirons votre attention sur le fait que chaque client devrait normalement être utilisable avec n'importe quel serveur développé par un autre binôme. Il convient pour cela d'appliquer scrupuleusement le protocole défini ici et de traiter convenablement les éventuels messages ou réponses non définis.

Cas d'erreurs

En cas d'erreur de syntaxe (en TCP), le serveur répondra :

```
{
   "statut":"400",
   "message": "Bad request"
}
```

En cas d'erreur inattendue ou non connue du système :

```
{
    "statut":"520",
    "message": "Unknown Error"
}
```

Recherche d'un serveur (UDP)

Message envoyé sur le réseau par le client en quête d'un serveur (broadcast UDP sur le port 5555) :

looking for quiznet servers

Message retourné par un serveur en réponse à la requête précédente (UDP) :

```
hello i'm a quiznet server:SERVER_NAME:PORT_TCP
```

où **SERVER_NAME** est le nom du serveur et **PORT_TCP** est le port TCP sur lequel le client devra se connecter.

Si le serveur reçoit un message ne correspondant pas à celui attendu, il ne répondra pas.

Inscription d'un joueur

Requête envoyée par le client (TCP) :

```
POST player/register
{
    "pseudo":"Player1",
    "password":"mypassword123"
}
```

Réponse retournée par le serveur en cas de succès :

```
{
    "action":"player/register",
    "statut":"201",
    "message":"player registered successfully"
}
```

En cas d'échec (pseudo déjà pris) :

```
{
    "action":"player/register",
    "statut":"409",
    "message":"pseudo already exists"
}
```

Connexion d'un joueur

Requête envoyée par le client :

```
POST player/login
{
    "pseudo":"Player1",
    "password":"mypassword123"
}
```

Réponse retournée par le serveur en cas de succès :

```
{
    "action":"player/login",
    "statut":"200",
    "message":"login successful"
}
```

En cas d'échec (mauvais identifiants) :

```
{
    "action":"player/login",
    "statut":"401",
    "message":"invalid credentials"
}
```

Récupération de la liste des thèmes

Requête envoyée par le client :

GET themes/list

Réponse retournée par le serveur :

```
{
    "action":"themes/list",
    "statut":"200",
    "message":"ok",
    "nbThemes":3,
    "themes":[
        {
            "id":0,
            "name":"Culture Générale"
        },
        {
            "id":1,
            "name":"Informatique"
        },
        {
            "id":2,
            "name":"Histoire"
        }
    ]
}
```


Récupération de la liste des sessions

Requête envoyée par le client :

GET sessions/list

Réponse retournée par le serveur :

```
{
    "action":"sessions/list",
    "statut":"200",
    "message":"ok",
    "nbSessions":2,
    "sessions":[
        {
            "id":0,
            "name":"Session du vendredi",
            "themeIds":[0,2],
            "themeNames":["Culture Générale","Histoire"],
            "difficulty":"moyen",
            "nbQuestions":20,
            "timeLimit":30,
            "mode":"solo",
            "nbPlayers":3,
            "maxPlayers":8,
            "status":"waiting"
        },
        {
            "id":1,
            "name":"Battle Informatique",
            "themeIds":[1],
            "themeNames":["Informatique"],
            "difficulty":"difficile",
            "nbQuestions":15,
            "timeLimit":25,
            "mode":"battle",
            "nbPlayers":2,
            "maxPlayers":10,
            "status":"waiting"
        }
    ]
}
```

avec :

- **status** : "waiting" (en attente de joueurs), "playing" (en cours), "finished" (terminée)
- **difficulty** : "facile", "moyen", ou "difficile"
- **timeLimit** : temps de réponse maximal en secondes pour chaque question

S'il n'existe pas de session en attente, le serveur retourne :

```
{
    "action":"sessions/list",
    "statut":"200",
    "message":"ok",
    "nbSessions":0
}
```

Création d'une session

Message envoyé par le client :

```
POST session/create
{
    "name":"Ma session",
    "themeIds":[1,2],
    "difficulty":"moyen",
    "nbQuestions":20,
    "timeLimit":30,
    "mode":"solo",
    "maxPlayers":6
}
```

Avec :

- **themeIds** : tableau d'identifiants de thèmes (au moins un)
- **difficulty** : "facile", "moyen", ou "difficile"
- **nbQuestions** : entre 10 et 50
- **timeLimit** : temps en secondes (entre 10 et 60)
- **mode** : "solo" ou "battle"

Réponse du serveur :

```
{
    "action":"session/create",
    "statut":"201",
    "message":"session created",
    "sessionId":5,
    "isCreator":true,
    "lives":4,
    "jokers":{
        "fifty":1,
        "skip":1
    }
}
```

Le champ **lives** n'est présent que pour les sessions en mode Battle.
Le créateur de la session peut la démarrer quand il le souhaite (avec au moins 2 joueurs).
En cas d'erreur (pas assez de questions correspondant aux critères) :

```
{
    "action":"session/create",
    "statut":"400",
    "message":"not enough questions matching criteria"
}
```

Rejoindre une session

Pour rejoindre une session, le client envoie la requête suivante :

```
POST session/join
{
    "sessionId":5
}
```

Le serveur répond :

```
{
    "action":"session/join",
    "statut":"201",
    "message":"session joined",
    "sessionId":5,
    "mode":"battle",
    "isCreator":false,
    "players":["Player1","Player2","Player3"],
    "lives":4,
    "jokers":{
        "fifty":1,
        "skip":1
    }
}
```

Le champ **lives** n'est présent que pour les sessions en mode Battle.
Le serveur envoie également à tous les autres joueurs de la session :

```
POST session/player/joined
{
    "pseudo":"Player3",
    "nbPlayers":3
}
```

En cas de session complète :

```
{
    "action":"session/join",
    "statut":"403",
    "message":"session is full"
}
```

Démarrage d'une session

Seul le créateur peut démarrer la session. Il envoie :

POST session/start

Le serveur vérifie qu'il y a au moins 2 joueurs, puis envoie à tous les joueurs :

```
POST session/started
{
    "message":"session is starting",
    "countdown":3
}
```

Après le countdown, le serveur envoie la première question.

Envoi d'une question

Le serveur envoie à tous les joueurs de la session :
Pour une question QCM :

```
POST question/new
{
    "questionNum":1,
    "totalQuestions":20,
    "type":"qcm",
    "difficulty":"moyen",
    "question":"Quel langage est utilisé côté serveur ?",
    "answers":["Python","C","JavaScript","Ruby"]
}
```

Pour une question Vrai/Faux :

```
POST question/new
{
    "questionNum":2,
    "totalQuestions":20,
    "type":"boolean",
    "difficulty":"facile",
    "question":"TCP est un protocole orienté connexion"
}
```

Pour une question Texte libre :

```
POST question/new
{
    "questionNum":3,
    "totalQuestions":20,
    "type":"text",
    "difficulty":"difficile",
    "question":"Quelle est la capitale de la France ?"
}
```

Notez que le temps limite et les points ne sont pas envoyés car ils sont déterminés par les paramètres de la session (timeLimit) et par la difficulté de la question.

Utilisation d'un joker

Le joueur peut utiliser un joker avant de répondre :

```
POST joker/use
{
    "type":"fifty"
}
```

Le serveur répond (pour le joker 50/50 sur une question QCM) :

```
{
    "action":"joker/use",
    "statut":"200",
    "message":"joker activated",
    "remainingAnswers":["C","JavaScript"],
    "jokers":{
        "fifty":0,
        "skip":1
    }
}
```

Pour le joker "Passer" :

```
{
    "action":"joker/use",
    "statut":"200",
    "message":"question skipped",
    "jokers":{
        "fifty":1,
        "skip":0
    }
}
```

Dans ce dernier cas, le serveur considère que le joueur a répondu et passe à la question suivante pour lui. En mode Battle, le joueur ne perd pas de vie.

Réponse à une question

Le joueur envoie sa réponse :
Pour un QCM (index de la réponse, 0-3) :

```
POST question/answer
{
    "answer":1,
    "responseTime":12.5
}
```

Pour un Vrai/Faux :

```
POST question/answer
{
    "answer":true,
    "responseTime":8.2
}
```

Pour un Texte libre :

```
POST question/answer
{
    "answer":"Paris",
    "responseTime":5.7
}
```

Le serveur accuse réception (sans révéler la bonne réponse) :

```
{
    "action":"question/answer",
    "statut":"200",
    "message":"answer received"
}
```

Résultats d'une question

Lorsque tous les joueurs ont répondu ou que le temps est écoulé, le serveur envoie à tous les joueurs :

En mode Solo :

```
POST question/results
{
    "correctAnswer":1,
    "explanation":"Le serveur doit être écrit en C ou C++",
    "results":[
        {
            "pseudo":"Player1",
            "answer":1,
            "correct":true,
            "points":13,
            "totalScore":215
        },
        {
            "pseudo":"Player2",
            "answer":0,
            "correct":false,
            "points":0,
            "totalScore":180
        },
        {
            "pseudo":"Player3",
            "answer":-1,
            "correct":false,
            "points":0,
            "totalScore":165
        }
    ]
}
```

Avec :

- **answer** :-1 signifiant "pas de réponse"
- **points** : les points gagnés pour cette question (incluant le bonus de rapidité si applicable)

Dans l'exemple ci-dessus, Player1 a répondu correctement à une question moyenne (10 points) dans la première moitié du temps (+3 points de bonus), pour un total de 13 points.

En mode Battle :

```
POST question/results
{
    "correctAnswer":1,
    "explanation":"Le serveur doit être écrit en C ou C++",
    "lastPlayer":"Player3",
    "results":[
        {
            "pseudo":"Player1",
            "answer":1,
            "correct":true,
            "responseTime":8.5,
            "points":13,
            "totalScore":215,
            "lives":4
        },
        {
            "pseudo":"Player2",
            "answer":0,
            "correct":false,
            "responseTime":12.3,
            "points":0,
            "totalScore":180,
            "lives":3
        },
        {
            "pseudo":"Player3",
            "answer":1,
            "correct":true,
            "responseTime":18.7,
            "points":10,
            "totalScore":175,
            "lives":3
        }
    ]
}
```

En mode Battle :

- **lastPlayer** : le pseudo du joueur ayant répondu en dernier (perd une vie)
- **lives** : nombre de vies restantes pour chaque joueur
- Les joueurs ayant répondu incorrectement perdent une vie
- Le joueur ayant répondu en dernier (même avec une réponse correcte) perd une vie
- Le score est calculé mais n'affecte pas le classement

Si un joueur n'a plus de vie, le serveur envoie également :

```
POST session/player/eliminated
{
    "pseudo":"Player2"
}
```

Fin de session

Après la dernière question, le serveur envoie le classement final :
En mode Solo :

```
POST session/finished
{
    "mode":"solo",
    "ranking":[
        {
            "rank":1,
            "pseudo":"Player1",
            "score":850,
            "correctAnswers":18
        },
        {
            "rank":2,
            "pseudo":"Player2",
            "score":520,
            "correctAnswers":15
        },
        {
            "rank":3,
            "pseudo":"Player3",
            "score":440,
            "correctAnswers":14
        }
    ]
}
```

En mode Battle :

```
POST session/finished
{
    "mode":"battle",
    "winner":"Player1",
    "ranking":[
        {
            "rank":1,
            "pseudo":"Player1",
            "lives":2,
            "score":850,
            "correctAnswers":18
        },
        {
            "rank":2,
            "pseudo":"Player3",
            "lives":0,
            "score":440,
            "correctAnswers":14,
            "eliminatedAt":15
        },
        {
            "rank":3,
            "pseudo":"Player2",
            "lives":0,
            "score":520,
            "correctAnswers":15,
            "eliminatedAt":12
        }
    ]
}
```

En mode Battle, le classement est basé sur :

1. Le nombre de vies restantes (le plus élevé gagne)
2. En cas d'égalité, le moment de l'élimination (**eliminatedAt** : numéro de la question où le joueur a été éliminé, le plus tard est le mieux)
3. Le score peut être affiché à titre informatif mais ne détermine pas le classement

Déconnexion d'un joueur

Si un joueur se déconnecte pendant une session, le serveur envoie à tous les autres joueurs :

```
POST session/player/left
{
    "pseudo":"Player2",
    "reason":"disconnected"
}
```

Le joueur déconnecté est considéré comme forfait.

Points d'attention particuliers

Synchronisation

La synchronisation des timers entre tous les clients est cruciale. Le serveur fait autorité sur le temps restant. Les clients doivent gérer l'affichage d'un timer local mais respecter le timeout

du serveur.

Gestion de la triche

Vous devez prévoir des mécanismes pour éviter la triche évidente :

- Vérifier que les réponses arrivent dans le délai imparti (côté serveur)
- Vérifier que le joueur ne répond qu'une seule fois par question
- Vérifier que les jokers utilisés sont bien disponibles
- Le temps de réponse doit être vérifié côté serveur

Robustesse

Votre système doit gérer correctement :

- La déconnexion brutale d'un client
- La déconnexion du créateur de session (un autre joueur devient créateur)
- Les réponses en double
- Les tentatives de rejoindre une session inexistante
- Les tentatives d'utiliser un joker non disponible

Persistance des données

Les données suivantes doivent persister (fichier, base de données simple) :

- Comptes joueurs (pseudo, mot de passe haché)
- Thèmes et questions

Sélection des questions

Lors de la création d'une session, le serveur doit :

- Sélectionner aléatoirement des questions correspondant aux thèmes et au niveau de difficulté choisis
- S'assurer qu'il y a suffisamment de questions disponibles
- Éviter de poser deux fois la même question dans une session

Gestion des vies en mode Battle

Le mode Battle nécessite une attention particulière :

- Chaque joueur commence avec 4 vies
- Une vie est perdue si le joueur répond incorrectement
- Une vie est également perdue si le joueur répond en dernier (même si la réponse est correcte)
- Un joueur avec 0 vie est éliminé mais peut rester connecté pour observer la fin de la partie
- La partie se termine quand il ne reste qu'un seul joueur avec des vies, ou à la fin des questions
- En cas d'égalité (plusieurs joueurs avec des vies à la fin), celui qui s'est fait éliminer le plus tard gagne

Extensions possibles (bonus)

Si vous terminez l'implémentation de base, voici quelques extensions intéressantes :

- **Chat textuel** : permettre aux joueurs de discuter dans la salle d'attente
- **Blind test** : intégrer un mode blind test musical ou visuel où les joueurs doivent reconnaître des extraits sonores ou des images
- **Création de questions** : permettre aux joueurs de soumettre leurs propres questions (validation par admin)
- **Classement global** : top 10 des meilleurs joueurs sur le serveur
- **Replay** : possibilité de revoir les réponses d'une session terminée
- **Questions avec images** : support d'images dans les questions
- **Difficulté mixte** : permettre de créer un quiz mélangeant différents niveaux de difficulté
- **Statistiques joueur** : réintégrer un système de statistiques pour suivre la progression des joueurs
- **Mode survie** : en mode Battle, on pourrait imaginer un système où les joueurs gagnent une vie bonus tous les X questions correctes

Y'a plus qu'à !

Nous avons décrit de manière assez précise l'ensemble des éléments composant ce projet. Vous allez cependant vous rendre compte qu'il subsiste quelques points non expliqués ici, comme par exemple la gestion exacte des cas d'égalité en mode Battle (que se passe-t-il si plusieurs joueurs répondent exactement au même moment ?), le format exact de stockage des questions, ou encore le mécanisme de gestion des timers. C'est à vous de trouver des solutions et définir vos propres extensions du protocole afin de traiter l'ensemble de nos attentes. Il se peut également que certaines propositions décrites ici ne soient pas adaptées à une utilisation normale de l'application (exemple : la gestion des jokers qui peut être un frein à la fluidité du jeu). N'hésitez pas, dans ce cas, à adapter les mecanismes pour faire de ce projet le petit bijou attendu... Gardez également en tête que ces choix et leur mise en œuvre ne devront pas, quoi qu'il en soit, impacter la compatibilité de vos client/serveur avec ceux des autres binômes...

Y'a plus qu'à donc.... et bon courage.