



UNIVERSITÉ  
DE LORRAINE

UNIVERSITÉ DE LORRAINE  
UFR MIM

UE RESEAU  
RAPPORT

---

## Projet Réseau : Programmation Socket

---

*Élève :*  
OTMANI MAMAR  
ZILLE LILIAN

*Enseignant :*  
LE HOAI MINH

## Table des matières

<b>1 Présentation générale</b>	<b>2</b>
1.1 Rappel du sujet . . . . .	2
1.2 Les langages de programmation . . . . .	2
1.2.1 Le serveur . . . . .	2
1.2.2 Le client . . . . .	2
<b>2 Le serveur et le client</b>	<b>3</b>
2.1 Quel service est rendu par le serveur ? . . . . .	3
2.2 Que demande le client ? . . . . .	3
2.3 Que répondra le serveur à la demande du client ? . . . . .	3
2.4 Qu'est-ce qui est donc à la charge du serveur ? . . . . .	3
2.5 Qui du serveur ou du client fixe les règles du jeu ? . . . . .	3
2.6 Définir la structure et les champs d'un message envoyé par le client . . . . .	4
2.7 Définir la structure et les champs d'un message envoyé par le serveur . . . . .	4
2.8 Elaborer la séquencement des messages échangés entre le client et le serveur	4
2.9 Définir précisément les diagrammes qui décrivent : une partie gagnée, perdue, abandonnée par le client, . . . . .	5
<b>3 Nos extensions</b>	<b>6</b>
3.1 Jeu contre le serveur . . . . .	6
3.2 Jeu avec limite temps . . . . .	7
3.3 Changement de difficultés . . . . .	7
3.4 Dérivé géographique . . . . .	7
3.5 Recommencer une partie . . . . .	7
3.6 Les modes multi-clients . . . . .	7
3.6.1 Les groupes . . . . .	7
3.6.2 Le Versus . . . . .	8
3.6.3 La coopération . . . . .	8
<b>4 Annexe : Quelques images du jeu</b>	<b>9</b>

# 1 Présentation générale

## 1.1 Rappel du sujet

Dans le cadre de l'UE Réseau, nous avons dû réaliser en binôme le jeu du pendu avec programmation socket. Dans un premier temps, la tâche était principalement de mettre en place un serveur afin qu'un client puisse se connecter à ce dernier et jouer une partie. Dans un second temps, il nous a été demandé de réaliser des extensions du jeu aussi diverses soient elles tout en conservant la structure initiale de notre serveur.

Toutes ces communications devaient se faire en usant de la programmation sockets. Il s'agit d'un modèle permettant la communication entre plusieurs processus afin qu'ils puissent communiquer aussi bien sur une même machine qu'à travers un réseau TCP/IP.

## 1.2 Les langages de programmation

Compte tenu du fait que le choix du langage de programmation était libre, nous avons décidé d'expérimenter une communication entre différents langages.

### 1.2.1 Le serveur

Afin de réaliser le serveur, nous avions besoin d'un langage permettant le traitement de données en back-end tout étant compatible avec la programmation sockets. Notre choix s'est donc porté sur le langage Python. D'une part pour sa facilité d'utilisation et d'autre part par simple curiosité. En effet, python étant un langage extrêmement populaire, nous ne l'avons que très peu pratiqué lors de notre cursus universitaire. De plus, Python est muni d'une librairie focalisée sur la programmation sockets. Cette dernière, nommée WebSockets, nous a permis de réaliser un serveur performant.

### 1.2.2 Le client

En ce qui concerne le client, nous voulions réaliser le jeu sous forme d'un site internet avec une interface graphique. Python permet la réalisation de cette tâche, cependant nous avons préféré utiliser le langage Java Script pour la partie client, plus précisément le framework Next.js, un framework gratuit et open source s'appuyant sur la bibliothèque JavaScript React et sur la technologie Node.js. Les raisons de ce choix furent, tout d'abord, pour l'utilisation des feuilles de style en cascade CSS permettant une plus grande personnalisation de l'interface, mais également pour le côté dynamique du langage JS. De plus, nous avons déjà eu l'occasion de pratiquer la programmation avec le framework Next par le passé.

Enfin, il existe également une librairie pour le framework Next permettant d'assurer des communications par le biais de websockets. Notre client pouvait donc communiquer avec notre serveur malgré les différents langages utilisés.

## 2 Le serveur et le client

Dans un premier temps, nous avons dû définir les règles du jeu. Pour cela nous avons imaginé le déroulé exact d'une partie dans un document texte afin d'établir un protocole de dialogue entre le client et le serveur. Une fois ce dernier réalisé, nous avons pu le traduire directement en code. A noter que dès le départ, notre protocole était pensé de telle sorte à ce que le serveur puisse accueillir de multiples clients sans pour autant risquer des conflits lors de l'envoie et là réception de messages.

Le protocole a pu être réalisé en se basant sur les questions présentes sur le sujet. Notons que les réponses à ces questions ne concernent que la première partie du projet, c'est à dire la réalisation d'une partie simple entre un client et un serveur :

### 2.1 Quel service est rendu par le serveur ?

Lors d'une partie, le serveur joue un rôle de maître du jeu. Il s'occupe d'initialiser une partie, c'est à dire sélectionner le mot qui sera à deviner et faire parvenir au client le mot caché, mais il gère également tout le déroulé de la partie.

### 2.2 Que demande le client ?

Comme expliqué plus haut, le client n'aura qu'à demander au serveur si la lettre qu'il propose est dans le mot ou non. Il peut également proposer un mot entier et demander au serveur s'il s'agit du mot à deviner ou non. Pour ne pas envoyer de requête inutile au serveur, le client vérifie si sa proposition ne contient de caractères spéciaux. De plus, la proposition peut contenir des lettres majuscules et minuscules, le serveur pourra tout de même l'interpréter.

### 2.3 Que répondra le serveur à la demande du client ?

Le serveur lui répondra alors si sa proposition est correcte ou non, tout en lui transmettant le mot caché actualisé (c'est à dire avec les lettres correctes dévoilées).

### 2.4 Qu'est-ce qui est donc à la charge du serveur ?

Le serveur est donc en charge du bon déroulement de la partie en passant par la réception des propositions du client, la vérification de la validité de la proposition, la mise au courant du client de si cette dernière est bonne ou fausse ,la gestion des erreurs et enfin la fin de partie.

### 2.5 Qui du serveur ou du client fixe les règles du jeu ?

Au vu du fait que le serveur est en charge du bon déroulement de la partie, c'est lui qui est en charge de fixer les règles du jeu. Parmi ces règles on retrouve par exemple le nombre d'erreur permis par le client.

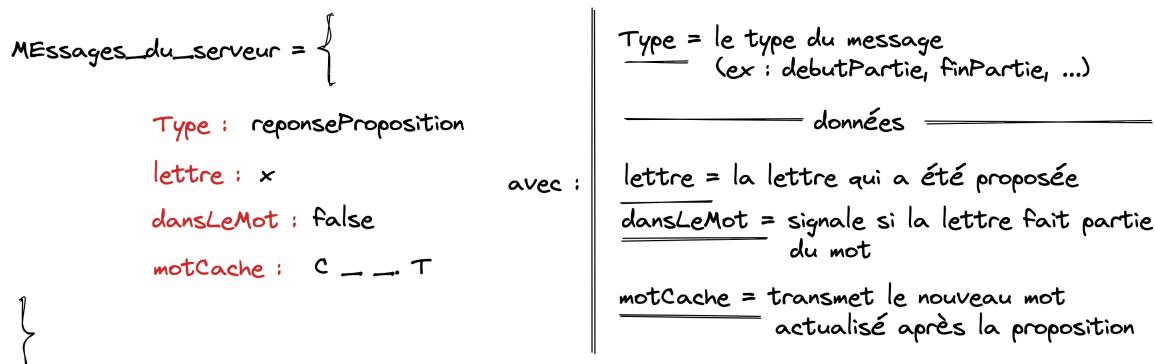
## 2.6 Définir la structure et les champs d'un message envoyé par le client

Nous avions initialement pensé à uniquement envoyer un message simple de type String afin que le client puisse communiquer avec le serveur. Au vu du fait que le client n'est capable que de proposer une lettre ou un mot, ce format semblait suffisant. Le serveur n'avait qu'à récupérer la chaîne de caractère et vérifier :

- Si elle est composée d'un seul caractère : si elle fait partie du mot.
- Si elle est composée de plusieurs caractères : si elle est égale au mot.

## 2.7 Définir la structure et les champs d'un message envoyé par le serveur

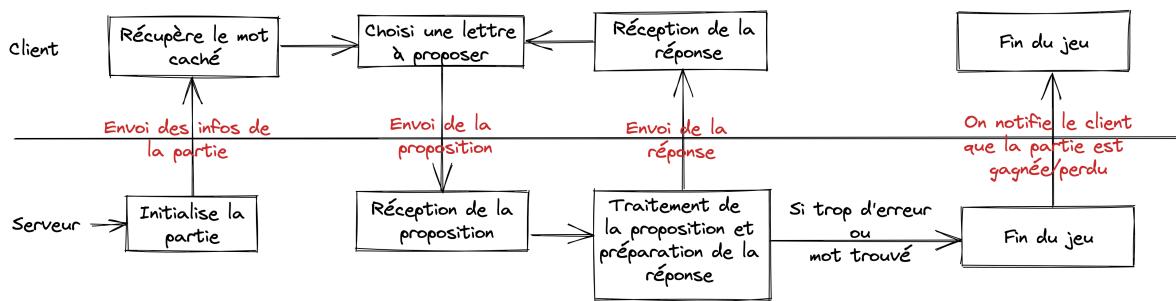
En ce qui concerne le serveur, il enverra plusieurs types de messages au client : Cela peut être la réponse d'une proposition du client, l'envoi du mot masqué, l'envoi de fin de partie en cas de défaite ou de victoire. Nous avons donc décidé de mettre en place des messages sous format JSON composé d'un type et des données nécessaires. Cette solution permet également de transmettre diverses informations grâce à un seul message, plutôt que d'en faire plusieurs.



Par la suite, ce format sera également adopté pour l'envoi des messages par le client afin de mieux gérer les extensions de la deuxième partie. En effet ce dernier devra envoyer des messages au server afin de choisir le mode de jeu, inviter dans son groupe, accepter des invitations, etc.

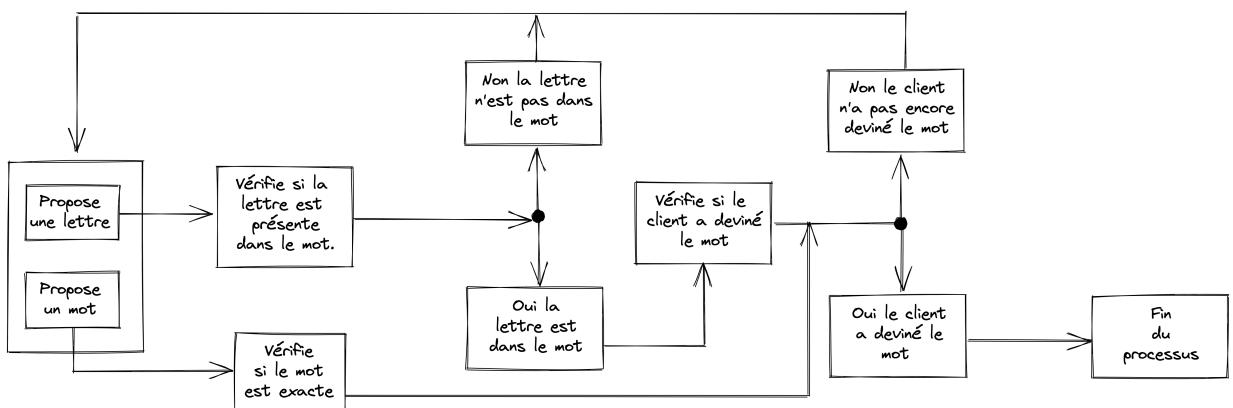
## 2.8 Elaborer la séquencement des messages échangés entre le client et le serveur

Ainsi, à partir de toutes les questions précédentes, nous avons été capable de réaliser un séquencement des messages échangés entre le client et le serveur lors du déroulement d'une partie :

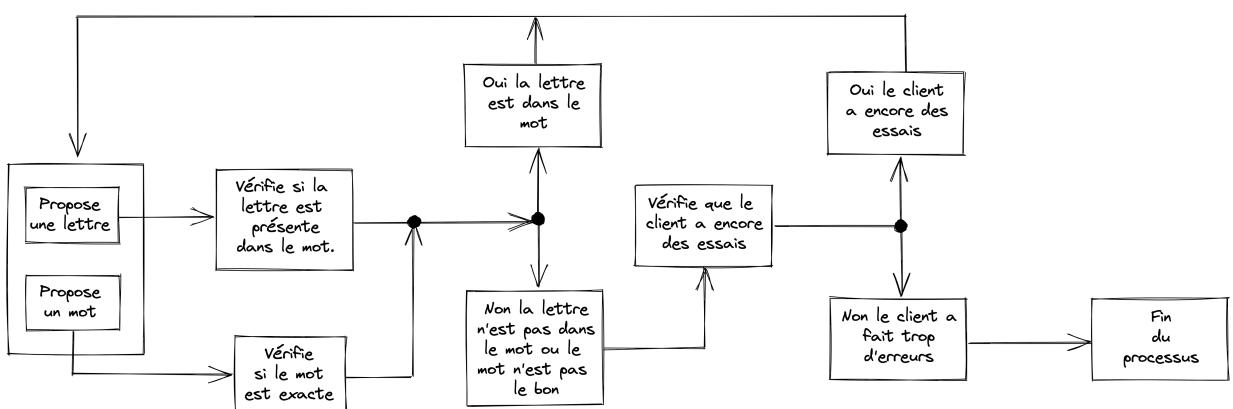


2.9 Définir précisément les diagrammes qui décrivent : une partie gagnée, perdue, abandonnée par le client, ...

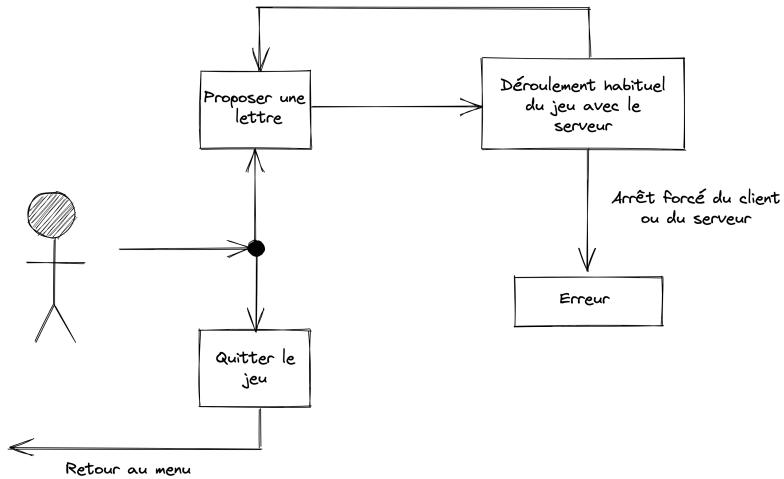
Dans le cas d'une partie gagnée :



Dans le cas d'une partie perdue :



Dans le cas d'un abandon de partie :



## 3 Nos extensions

Pour la deuxième partie du rapport, il nous a été demandé d'ajouter diverses extensions à notre projet. Au total nous avons ajouté 5 différentes extensions :

### 3.1 Jeu contre le serveur

Le jeu contre le serveur est un mode de jeu additionnel dans lequel le principe du jeu du pendu est inversé. Ici, c'est le client qui propose un mot et le serveur a pour mission de le deviner. Cette tâche peut sembler nécessiter une partie d'intelligence artificielle, cependant nous avons pu arriver à un résultat sans utiliser d'IA :

Tout d'abord, nous avons récupéré une liste de mots présents dans le dictionnaire français en format txt trouvable sur internet. Cette liste contient 386264 mots.

Lorsque le client propose un mot, le serveur va tout d'abord garder de côté tous les mots issus de la liste précédemment mise en place dont le nombre de lettres correspond au nombre de lettres du mot du client.

Suite à cela, le serveur va commencer à tester les lettres de l'alphabet une par une par ordre de fréquence comme le ferait une personne jouant au pendu (la plupart des personnes commencent par les voyelles E, A et I).

Pour chaque lettre, le serveur va tout d'abord vérifier si elle est contenue dans au moins un mot parmi ceux qu'il a gardé en stock précédemment. Si c'est le cas alors il la teste sur le mot que le client a donné. Deux cas distincts peuvent survenir à ce moment :

- Si la lettre est présente dans le mot mystère, alors le serveur va éliminer tous les mots qui ne contiennent pas cette lettre de ses mots en stock.

- Au contraire, si la lettre n'est pas présente dans le mot mystère, alors le serveur va éliminer tous les mots qui contiennent cette lettre dans ses mots en stock.

De cette façon, le nombre de mots pouvant être le mot mystère diminue grandement. Lorsque ce nombre est suffisamment bas, le serveur va cesser de proposer des lettres dans

l'ordre de fréquence et va se concentrer sur les lettres présentent dans les derniers mots restants.

### 3.2 Jeu avec limite temps

Un autre mode de jeu est un mode dans lequel le joueur devra trouver le mot caché sans faire trop d'erreurs mais également avant la fin d'un chrono. Ce chrono est essentiellement géré par le client notamment pour des questions d'affichage puisque qu'il a été compliqué de synchroniser le temps qui s'écoule au niveau du client et du serveur. Si nous voulions qu'un seul chrono il aurait fallu que le serveur tienne à jour le client toutes les secondes de l'avancé du temps, ce qui aurait pu créer des conflits si le client voulait proposer une lettre au même moment.

### 3.3 Changement de difficultés

Le jeu permet de changer la difficulté. Nous avons mis en place trois niveaux de difficultés différents : Le niveau facile où le joueur a droit à 10 vies et à 60 secondes dans le mode de jeu avec limite de temps, le niveau moyen où il a droit à 8 vies et 40 secondes, et enfin le niveau difficile où cette fois ci, ce seront 5 vies et 20 secondes.

### 3.4 Dérivé géographique

Voici une autre extension du jeu du pendu. Cette fois-ci, l'objectif principal n'est pas de deviner un mot caché mais un pays ou une capitale. Ce mode ayant pour but de faire appel à la culture générale du joueur, au fur et à mesure que le nombre d'erreurs augmente, nous lui proposons divers indices afin de l'aider à trouver la bonne réponse. Le premier indice étant le type du mot à trouver (s'il s'agit d'un pays ou d'une capitale). Le deuxième indice se trouve être le continent où se situe le lieu mystère. Enfin, le troisième indice est, dans le cas d'un pays : sa capitale, et inversement.

### 3.5 Recommencer une partie

Une des extensions proposées par le sujet était de recommencer une partie rapidement. Ainsi, lorsqu'une partie arrive à son terme, un écran de fin apparaît (Felicitation en cas de victoire, Game over en cas de défaite) laissant le choix au joueur de relancer directement une partie avec les mêmes paramètres de difficultés ou retourner au menu principal.

### 3.6 Les modes multi-clients

Enfin, la dernière extension est l'ajout d'un mode de jeu multijoueur. Ce mode a été conduit à l'implémentation de plusieurs nouvelles fonctionnalités :

#### 3.6.1 Les groupes

Lorsqu'un client est connecté au serveur, il a accès à la liste de tous les autres clients actuellement sur le serveur. Il a ainsi la possibilité de l'ajouter dans son groupe ou de se faire inviter dans le groupe de quelqu'un d'autre. Une fois dans le groupe, seul le chef a la

possibilité de sélectionner une difficulté ainsi qu'un mode de jeu. Tous les autres membres du groupes rejoindront automatiquement le mode de jeu qu'il aura sélectionné.

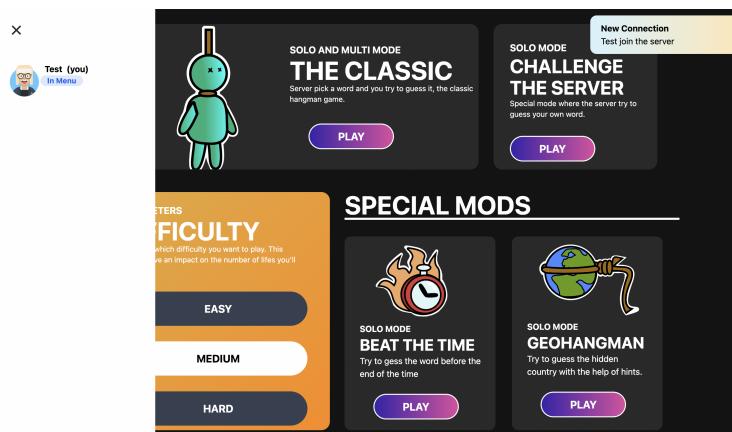
### 3.6.2 Le Versus

Le versus est un mode de jeu en multijoueur uniquement qui met en concurrence tous les joueurs d'un même groupe. Un unique mot mystère est tiré au sort et proposé à tous les joueurs. Le premier d'entre eux qui arrive à trouver le bon mot gagne la partie. Lorsque la partie est terminée, seul le chef aura le choix de recommencer la partie ou bien de retourner au menu.

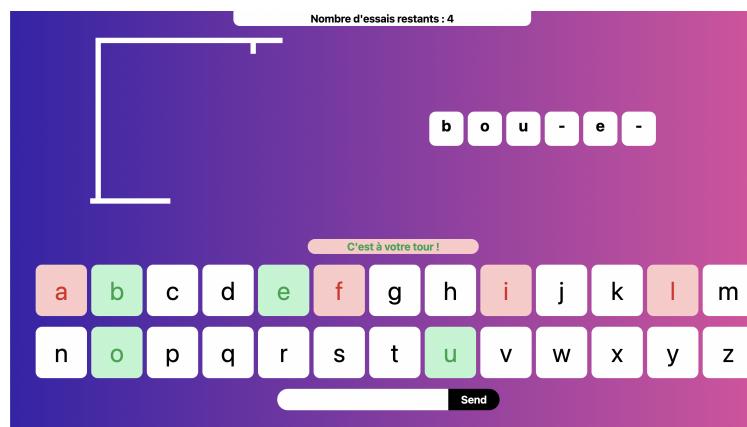
### 3.6.3 La coopération

La coopération est le mode de jeu inverse du versus. Ici, tous les joueurs d'un même groupe coopèrent afin de trouver le mot mystère. chacun d'en eux joue à tour de rôle et a le choix entre proposer une lettre ou proposer un mot entier. Attention cependant car le nombre d'erreurs est commun à tous les joueurs, ainsi lorsqu'un joueur perd, tous les joueurs perdent.

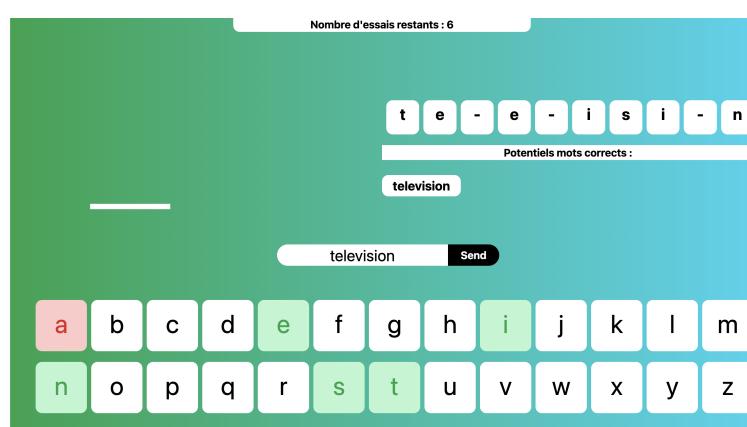
## 4 Annexe : Quelques images du jeu



*Menu principal du jeu*



*Ecran de jeu classique*



*Ecran de jeu où le serveur devine*