



Rapport de Soutenance Final

BELLONI Corentin COURTEMANCHE Jessy
CORCIONE Arnaud GIRARD Ethan
NGOUESSY BOUSSAMBA Daniel

22 mai 2023

Table des matières

1	Introduction	3
1.1	Présentation du projet	3
1.2	Présentation des membres	4
1.2.1	Arnaud Corcione	4
1.2.2	Corentin Belloni	4
1.2.3	Ethan Girard	4
1.2.4	Daniel Ngouessy	5
1.2.5	Jessy Courtemanche	5
1.3	Répartition des tâches	6
2	Avancement du projet	6
2.1	Serveur	6
2.1.1	Introduction	6
2.1.2	Protocole utilisé : TCP	6
2.1.3	Fonctionnement du Serveur	8
2.2	Serialization et Deserialization	13
2.2.1	Serialization	13
2.2.2	Deserialization	13
2.3	Client	16
2.3.1	Première soutenance	16
2.3.2	Deuxième soutenance	17
2.3.3	Soutenance finale	20
2.3.4	Parsing des réponses du serveur	21
2.3.5	Synchronisation globale du jeu	22
2.3.6	Changement majeur au niveau du client	23
2.4	Générateur de texte	25
2.4.1	Introduction	25
2.4.2	Commencement	25
2.4.3	Première partie	26
2.4.4	Seconde partie	27
2.4.5	Depuis la dernière soutenance	28
2.5	Affichage du jeu	32
2.5.1	Introduction	32
2.5.2	Code couleur	32
2.5.3	Fonctionnement	33
2.5.4	Statistiques du joueur	35
2.6	TUI	35
2.6.1	Pourquoi ncurses	35

2.6.2	Menu	36
2.6.3	Jeu	37
2.6.4	Aide	38
2.6.5	Quitter	39
2.7	Chat	40
2.7.1	Première soutenance	40
2.7.2	Deuxième soutenance	41
2.7.3	Objectif soutenance finale	43
2.7.4	Introduction des pthreads mutex	44
2.7.5	Introduction de Ncurses	46
2.8	Site web	49
2.9	Conclusion	50

1 Introduction

1.1 Présentation du projet

L'objectif de ce projet est de développer un jeu compétitif pour deux joueurs, qui consiste à taper le plus rapidement possible les suites de mots générées par l'application. Ce jeu vise à améliorer la rapidité et la précision de frappe des joueurs tout en offrant une expérience de jeu interactive et amusante.

Pour atteindre cet objectif, nous avons choisi de programmer en langage C et d'utiliser des bibliothèques et des outils appropriés pour créer une interface utilisateur attrayante et une expérience de jeu fluide. Le jeu sera doté d'un générateur de texte aléatoire pour créer des suites de mots pour les joueurs. Les joueurs devront taper les mots correctement et rapidement pour gagner des points.

Nous prévoyons d'implémenter des fonctionnalités pour suivre les statistiques des joueurs, tels que le nombre de victoires et le temps de réponse moyen. Nous nous efforçons de créer une expérience de jeu dynamique et engageante pour les joueurs, avec un niveau de difficulté croissant au fur et à mesure que le jeu progresse.

1.2 Présentation des membres

1.2.1 Arnaud Corcione

Depuis le début de ma jeunesse, grâce à mon père , j'ai été bercé dans le monde de l'informatique, ce qui m'a permis de m'intéresser à ce monde étrange. A mon entrée à Epita, j'ai eu une révélation pour le code, qui maintenant ne quitte pas mon quotidien. J'ai également pu réaliser des projets touchant à plusieurs langages. Ainsi, se creuser la tête pour apprendre et à comprendre une nouvelle technologie est passionnant pour moi.

1.2.2 Corentin Belloni

Je suis quelqu'un qui est passionné par l'informatique. J'aime découvrir de nouvelles technologies et mettre mes connaissances en pratique pour résoudre des problèmes complexes. Je suis également quelqu'un de joyeux, qui aime travailler en équipe et partager mes connaissances avec les autres. Je suis très investi dans mon travail et j'aime relever des défis. Je suis également très travailleur et je m'efforce toujours de donner le meilleur à moi-même.

1.2.3 Ethan Girard

Étant issu d'une terminale scientifique, j'ai été pris de passion par l'informatique en classe de première avec la spécialité "Numérique et sciences informatiques". Je suis quelqu'un qui aime l'informatique et le sport en général, je suis également passionné par les technologies et l'innovation. Je suis déterminé et engagé, prêt à me donner à fond pour créer un projet de A à Z.

1.2.4 Daniel Ngouessy

J'ai découvert cette passion qu'est l'informatique il y a quelques années et depuis, je m'efforce d'en apprendre toujours plus sur les dernières technologies et tendances dans ce domaine. Je suis également dynamique et motivé, ce qui me permet de m'investir pleinement dans tous les projets auxquels je participe. Je suis convaincu que mon enthousiasme et mon dévouement seront un atout pour réussir ce projet d'école.

1.2.5 Jessy Courtemanche

Depuis jeune, mes frères m'ont inculqué la passion du jeu vidéo, puis de fil en aiguille je me suis intéressé au domaine de l'informatique et chacun de mes centres d'intérêts me stimulaient d'avantage que les anciens. Il y a eu le hardware, le gaming que cela soit entre copains ou en mode compétition (l'eSport), mais aussi le software. La programmation m'a toujours particulièrement intéressé car c'était une énigme pour moi, et surtout je savais que tôt ou tard j'allais pouvoir développer mes compétences techniques dans ce domaine. A mon entrée à EPITA, j'avais seulement vaguement pratiqué le langage C pour utiliser Arduino ou bien des logiciels comme Flowcode permettant de contrôler des micro contrôleurs inclus dans des systèmes plus importants. Au-delà de ce que nous étudions en Sciences de l'Ingénieur, j'ai acheté un kit Arduino qui m'a permis de découvrir les bases du C et de la programmation en général. Mais tout ceci sans vraiment comprendre la logique camouflée derrière de tels outils. Je pense que notre projet va me faire découvrir de nouvelles choses. La programmation est pour moi une source constante d'apprentissage, le domaine étant tellement vaste.

1.3 Répartition des tâches

Tâches	Personne concernée
Serveur	Arnaud
Client	Daniel
Fonctionnement du jeu	Corentin
Générateur de texte	Ethan
TUI	Corentin
Chat	Jessy
Site Web	Jessy

2 Avancement du projet

2.1 Serveur

2.1.1 Introduction

Le serveur est un aspect important de notre jeu. Effectivement sans serveur, il n'y a pas de multijoueur. C'est pour cela que le développement de ce dernier n'est pas à négliger. Pour prouver que le jeu est un véritable multijoueur nous utilisons un serveur qui est hébergé dans le sud de la France, à La Ciotat. Arnaud étant responsable de ce serveur, il a donc logiquement développé tous le code pour le bon fonctionnement.

2.1.2 Protocole utilisé : TCP

TCP, Transmission Control Protocol, est un protocole de communication qui assure un transfert fiable et ordonné de données entre des ordinateurs sur un réseau. Ce protocole fonctionne en établissant une connexion entre l'expéditeur et le destinataire. Cette connexion se compose de trois étapes.

Tout d'abord, le client envoie une demande de connexion (SYN) au serveur. Le serveur répond avec un accusé de réception (SYN-ACK) pour confirmer la demande et établir la connexion. Enfin, le client envoie un accusé de réception (ACK) pour confirmer la réponse du serveur et finaliser la connexion.

Une fois la connexion établie, TCP divise les données à envoyer en segments plus petits. Chaque segment est numéroté et envoyé séparément. Le destinataire reçoit les segments, les confirme et renvoie un accusé de réception pour informer l'expéditeur que les données ont bien été reçues. Si le destinataire ne reçoit pas un segment, il demande à l'expéditeur de le renvoyer.

Le protocole TCP nous assure donc de la bonne réception des données envoyées entre le client et le serveur, ce qui est primordial. Car nous ne pouvons pas nous permettre de perdre des données sur le chemin. Les données échangées sont :

- Information de Win Condition
- Perte des points pour l'autre joueur
- Les différents messages pour le chat
- L'envoi d'une seed par le Serveur.
- Divers informations envoyés par le Serveur

Comme vous pouvez le voir, nous n'avons pas beaucoup de données à partager entre les différents clients, alors il est très important de s'assurer d'une bonne communication. C'est pour cela que le protocole TCP est adapté à ce qu'on souhaite faire.

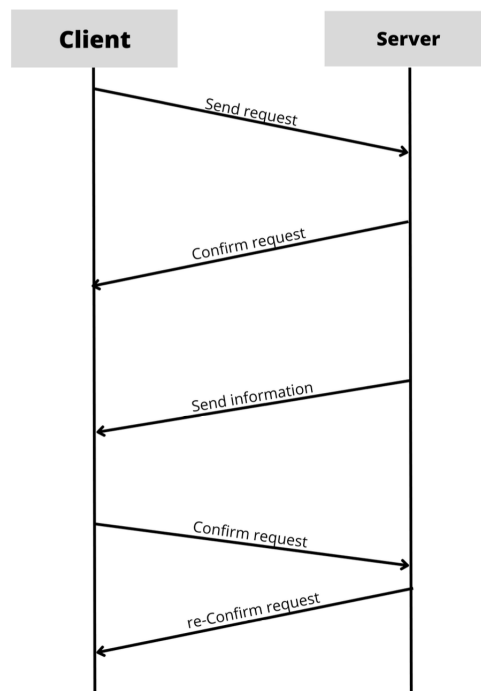


FIGURE 1 – Diagramme du Protocole TCP

2.1.3 Fonctionnement du Serveur

Dans un premier temps, il est important de noter que le serveur utilise plusieurs threads pour effectuer plusieurs tâches.

L'utilisation de threads dans le serveur présente en effet plusieurs avantages par rapport aux processus enfants. Tout d'abord, les threads sont plus légers que les processus et consomment moins de ressources système. Ils permettent également de partager plus facilement la mémoire et les ressources avec le processus parent, ce qui peut améliorer les performances globales du système.

De plus, Un thread est un processus capable d'exécuter du code en parallèle du programme principale. Ainsi il est possible de faire plusieurs actions, totalement différent en simultanée. Ce qui rend son utilisation puissante et très utile pour un serveur. Ce qui est particulièrement utile dans les systèmes multijoueurs.

Enfin, la communication entre les threads est simplifiée par rapport aux processus enfants. Comme les threads partagent la même mémoire, il est plus facile d'échanger des informations entre eux.

Actuellement le serveur est capable d'effectuer 2 principales actions en même temps :

- Gérer la connexion entre les différents client connecté.
- Lancer les différentes parties qui sont prêtes.

Lorsqu'un joueur se connecte plusieurs actions sont effectués du côté serveur.

Dans un premier temps, le serveur demande un nom au client, ce nom servira d'identifier les joueurs et surtout lors de la communication par le chat.

Une fois le nom donné, plusieurs actions s'enchaînent. Le joueur est ajouté à une room, ceci est comme un tunnel entre deux joueurs. Les deux peuvent ainsi échanger plusieurs informations dans ce tunnel sans interférer avec les autres joueurs déjà présent. Chaque room est isolé des autres. Ainsi le serveur fonctionne comme un relais, qui renvoie principalement les informations, à l'exception de quelques cas.

Pour résumer soit le client communique avec l'autre client présent, soit le serveur envoie une information aux deux clients, ce qui donne le diagramme suivant.

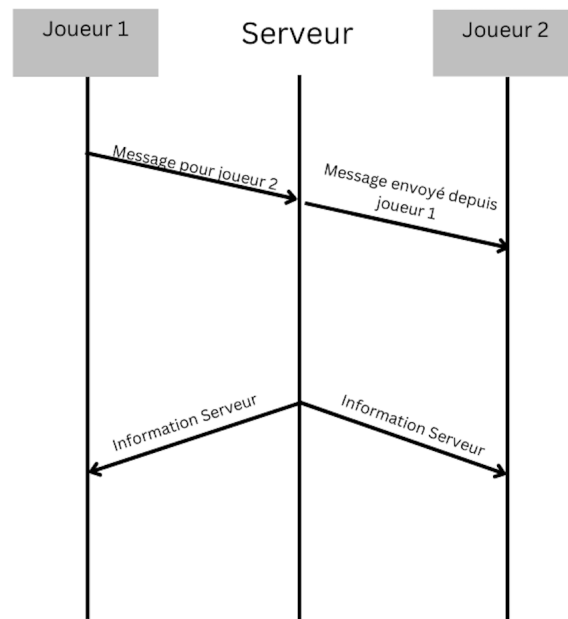


FIGURE 2 – Diagramme Communication Serveur - Clients

De plus, dès lors où les deux clients marquent le message "ready" alors ce message devient une commande serveur, ce qui met à jour les informations de la room pour se préparer à lancer le jeu. Évidemment tout au long de la connexion il est nécessaire d'informer le client sur les différentes choses que l'autre joueur peut faire, connexion, déconnexion, prêt à jouer.

```
Give me a name: Enzo
Arnaud is connected.
Enzo: Salut
Arnaud: Salut
Arnaud is ready to play.
Enzo:
```

FIGURE 3 – Image de la connexion d'un client

Ainsi quand les deux joueurs ont marqué "ready" et donc qu'ils sont prêts à jouer. Le serveur génère une Seed, un nombre grand afin d'avoir toujours le même aléatoire pour les différents clients, ce qui permet donc de générer les mêmes lignes pour les deux clients pour assurer de l'équité durant la partie. Évidemment, le serveur fait un décompte pour informer les joueurs du début de la partie.

```
Enzo: ready
Arnaud is ready to play.
Game start in 3
Game start in 2
Game start in 1
2071925510
```

FIGURE 4 – Lancement d'une partie.

Enfin pour finir, le serveur est capable d'envoyer plusieurs information comme par exemple, le fait qu'un client vienne de se connecter et averti le premier client déjà connecté. Ces derniers sont donc envoyé sous la forme suivant :

- "**Serveur** : Ceci est un message"
- "**Serveur Seed** : 7643789"

Plusieurs possibilités en fonction de l'action faite par les différents clients.

2.2 Serialization et Deserialization

2.2.1 Serialization

En effet, l'utilisation d'une structure générique est une pratique courante en sérialisation, car cela permet de traiter différents types d'objets ou de structures de données de manière polymorphe. En utilisant un pointeur vers une structure générique, on peut facilement parcourir et extraire les champs de données de n'importe quelle structure. Cela simplifie considérablement le processus de sérialisation, car il n'est pas nécessaire de créer une fonction de sérialisation différente pour chaque type de structure.

Cependant, il est important de noter que la sérialisation présente également des limites et des défis. De plus, la sérialisation peut également affecter les performances du système, en particulier lorsque les objets sérialisés sont volumineux ou doivent être envoyés à travers un réseau lent ou peu fiable. Il est donc important de bien comprendre les avantages et les inconvénients de la sérialisation, ainsi que de prendre en compte les aspects de performance et de sécurité lors de la mise en œuvre de cette méthode. Avec une conception et une implémentation appropriées, la sérialisation peut être un outil puissant pour faciliter l'échange de données dans diverses applications.

2.2.2 Deserialization

La désérialisation est une étape cruciale dans la communication entre des systèmes informatiques. Elle permet de convertir une représentation linéaire d'un objet ou d'une structure de données, généralement sous forme de chaîne de caractères ou de flux d'octets, en une forme utilisable par un langage de programmation.

Contrairement à la sérialisation, qui consiste à transformer une structure de données en une représentation linéaire, la désérialisation nécessite une analyse de cette représentation linéaire pour reconstruire la structure de données initiale.

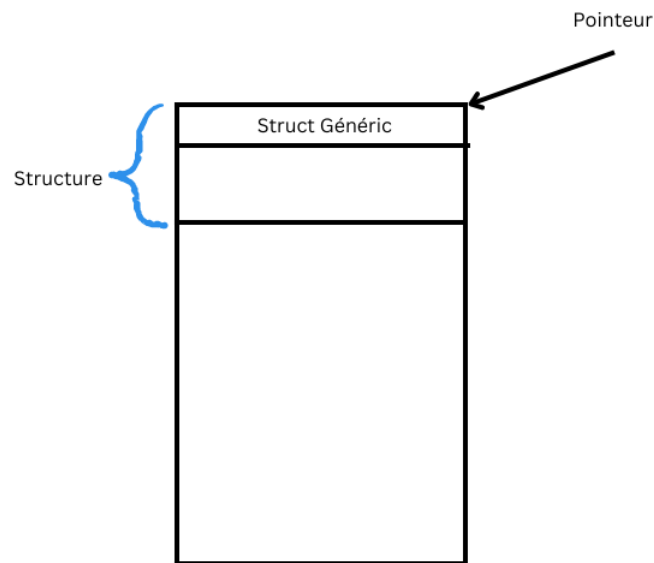


FIGURE 5 – Image de la Stack avec représentation d’une structure.

Pour effectuer la désérialisation, on utilise généralement un principe de structure générique. Cela permet de traiter des objets ou des structures de données de manière polymorphe, c'est-à-dire en utilisant une même méthode pour tous les types d'objets possibles. En analysant la chaîne de caractères ou le flux d'octets, on peut identifier les différents champs qui composent l'objet ou la structure de données, et les placer dans une nouvelle structure qui sera utilisée par le langage de programmation.

2.3 Client

2.3.1 Première soutenance

Lors de notre première soutenance de projet, nous avons discuté du rôle et des caractéristiques d'un client dans un système client-serveur. Un client est un logiciel informatique qui envoie des requêtes à un serveur. Il établit une connexion avec le serveur en spécifiant un ou plusieurs ports réseau, et une fois la connexion acceptée, il communique avec le serveur selon les exigences du protocole utilisé, tel que TCP.

Il est important que le client et le serveur utilisent le même protocole pour l'échange d'informations. Cet échange peut se faire soit sur le réseau, soit localement. Les clients et les serveurs doivent tous deux comprendre et respecter le protocole d'échange, dans notre cas, TCP.

Nous avons également discuté des différents types de clients applicatifs. Tout d'abord, il y a le client léger, où le traitement des requêtes est entièrement effectué par le serveur. Le client reçoit et affiche les réponses calculées par le serveur. Les avantages du client léger sont qu'il nécessite peu de puissance de calcul du côté du client, les mises à jour de l'application se font principalement sur le serveur, et il y a une plus grande indépendance entre les applications et les clients.

Ensuite, il y a le client lourd, où une partie importante du traitement est effectuée sur l'ordinateur local du client. Le serveur répond principalement aux demandes de données du client. Les avantages du client lourd incluent la possibilité de fonctionner même lorsque le serveur est déconnecté, le soulagement des ressources du serveur grâce au traitement effectué par le client, et une plus grande indépendance vis-à-vis du temps de réponse du réseau et du serveur.

Enfin, nous avons abordé le client riche, qui est une combinaison du client léger et du client lourd. Dans un client riche, le traitement des requêtes est principalement effectué par le serveur, mais le client finalise les réponses partiellement terminées. Cela permet d'avoir des fonctionnalités avancées similaires à celles d'un client lourd, tout en maintenant certains avantages d'un client léger.

Pour notre projet, nous avons décidé d'implémenter un client lourd dans un premier temps, mais cette décision est susceptible d'évoluer en fonction de nos besoins. L'implémentation prévue consiste à envoyer des requêtes au serveur et à recevoir des réponses en retour. Actuellement, nous avons fourni un manuel d'aide léger au client pour qu'il puisse connaître les commandes autorisées lors de la connexion.

En résumé, lors de notre première soutenance de projet, nous avons discuté du rôle et des caractéristiques d'un client dans un système client-serveur. Nous avons examiné les types de clients applicatifs, notamment le client léger, le client lourd et le client riche. Pour le moment, nous avons opté pour un client lourd dans notre implémentation, mais cela pourrait changer à l'avenir en fonction de nos besoins.

2.3.2 Deuxième soutenance

Lors de notre deuxième soutenance de projet, nous avons discuté de la mise en place de la connexion entre le client et le serveur dans notre jeu de saisie de phrases. Notre première étape a été d'ouvrir un socket côté serveur en utilisant la fonction `socket()`. Nous avons créé un socket et l'avons connecté à une adresse IP et à un port spécifiques. Ensuite, nous avons entré dans une boucle infinie pour écouter les connexions entrantes.

Lorsqu'un client se connecte, nous créons un nouveau socket côté serveur et acceptons la connexion du client, ce qui permet au serveur d'être prêt à communiquer avec le client.

La communication entre le client et le serveur s'effectue via des messages, et nous avons défini un protocole de communication pour structurer ces messages. Le client envoie des messages au serveur pour informer de son état de jeu et synchroniser les statistiques avec le serveur. Le serveur envoie des messages au client pour transmettre des informations sur le jeu, telles que le début d'une nouvelle manche ou la fin du jeu.

Nous avons choisi d'utiliser le protocole TCP/IP pour assurer une communication fiable entre le client et le serveur, garantissant que les messages sont reçus dans l'ordre et sans interruption. Nous avons défini la structure de nos messages en utilisant des structures de données en langage C. Chaque message comporte un en-tête contenant des informations sur le type de message et la longueur des données, ainsi que d'autres informations de contrôle. Les données elles-mêmes peuvent être de différents types, tels que des chaînes de caractères pour les phrases à taper ou des entiers pour les statistiques de jeu. Nous utilisons la sérialisation de données pour convertir ces structures de données en messages binaires qui peuvent être envoyés au serveur.

Le client envoie des messages au serveur pour informer de son état de jeu, tels que les statistiques de jeu lorsqu'une manche est terminée. Le serveur stocke ensuite ces données dans une base de données pour une analyse ultérieure. Lorsque le serveur reçoit un message, nous utilisons une fonction de désérialisation pour convertir les données binaires en une structure de données manipulable dans le code serveur.

Le serveur envoie également des messages au client pour transmettre des informations sur le jeu, comme le début d'une nouvelle manche ou la fin du jeu. Par exemple, lorsque tous les joueurs sont prêts à commencer une nouvelle manche, le serveur envoie un message à tous les clients pour les informer. Pour la synchronisation des statistiques, nous avons mis en place une structure de données côté client pour stocker les statistiques du joueur, qui sont ensuite envoyées au serveur via un message.

Nous avons également mis en place un système de synchronisation pour optimiser l'expérience de jeu, en créant un menu terminal permettant aux joueurs de choisir facilement le type de partie qu'ils souhaitent jouer, qu'il s'agisse d'une partie en ligne, d'un mode d'entraînement hors ligne ou d'un chat pour discuter avec d'autres joueurs.

En résumé, lors de notre deuxième soutenance de projet, nous avons mis en place la connexion entre le client et le serveur dans notre jeu de saisie de phrases. Nous avons utilisé un protocole de communication défini pour assurer une transmission efficace des données entre les deux parties.

Grâce à la sérialisation et à la désérialisation des données, nous avons pu convertir les structures de données en messages binaires et vice versa. Le client envoie des messages pour informer de son état de jeu et synchroniser les statistiques, tandis que le serveur envoie des messages pour transmettre des informations sur le jeu. Nous avons également créé un menu terminal pour une expérience de jeu fluide. Cependant à cette étape, les différentes parties du jeu fonctionnent encore de manière asynchrone.

2.3.3 Soutenance finale

Lors de notre soutenance finale, nous avons apporté des solutions à divers problèmes de synchronisation, de traitement des requêtes, et nous avons effectué une modification significative au niveau du client pour optimiser notre environnement de travail.

Grâce à nos efforts continus, nous avons pu améliorer la synchronisation globale du jeu, assurant ainsi une meilleure coordination entre les différentes parties. Nous avons résolu les problèmes d'asynchronisme qui existaient auparavant, en intégrant et en modifiant des éléments de code provenant des travaux des autres membres de l'équipe.

Un autre aspect crucial que nous avons pris en compte est le traitement efficace des requêtes entre le client et le serveur. Nous avons développé des mécanismes de parsing sophistiqués pour analyser et interpréter les réponses du serveur, garantissant ainsi une communication précise et fiable. En comprenant correctement les informations transmises par le serveur, nous avons pu adapter les actions du client en conséquence.

Enfin, l'un des changements majeurs que nous avons effectués concerne le client lui-même. Nous avons constaté que l'utilisation de deux clients distincts pour gérer le chat et le mode multijoueur était une approche inefficace. Pour remédier à cela, nous avons fusionné les fonctionnalités du client du chat et du client du multijoueur. Cette modification a permis une meilleure détection et un meilleur traitement des informations, en simplifiant le parsing et en facilitant la synchronisation.

Dans les sections suivantes, nous détaillerons plus précisément les solutions que nous avons mises en place pour résoudre ces problèmes et les améliorations apportées à notre jeu DigitDash. Nous sommes fiers des résultats obtenus lors de cette dernière étape de développement, et nous sommes impatients de partager ces avancées avec vous.

2.3.4 Parsing des réponses du serveur

Le bonne analyse d'une réponse du serveur par le client est essentielle pour le bon fonctionnement du jeu. Arnaud a donc mis en place un code permettant de discerner à quel type de réponse le client fait face. Notre travail a été de parser cette réponse du serveur comme suite :

Si le client reçoit comme réponse :

- **Serveur** : "Ceci est un message"
Nous affichons uniquement le message du serveur au client après avoir parsé ce-dernier.
- **Serveur seed** : "147851588"
Nous affichons uniquement la seed générée par ce-dernier.
Arnaud vous en parle plus en détails dans sa section 11

En effectuant cette analyse et en utilisant un processus de parsing, nous sommes en mesure de traiter les réponses du serveur de manière appropriée et de fournir au client des informations pertinentes. Cette étape est essentielle pour garantir une communication fluide entre le client et le serveur.

2.3.5 Synchronisation globale du jeu

Afin d'améliorer la synchronisation globale du jeu, nous avons identifié un problème d'asynchronisme entre les différentes parties du jeu, malgré la présence du Menu Principal. Pour remédier à cette situation, nous avons entrepris des modifications et des intégrations de code spécifique à chaque partie, en collaboration avec les autres membres de l'équipe.

Le Menu Principal est la porte d'entrée du jeu et permet également d'accéder au chat. Lorsqu'un joueur choisissait de rejoindre une partie en multijoueur, il était dirigé vers une salle de discussion où les joueurs pouvaient discuter avant de décider de lancer une partie. Cependant, nous avons rencontré des interférences entre la gestion du clavier par le chat et celle du jeu lui-même. De plus, il était impossible de lancer une partie directement depuis le chat.

Pour résoudre ces problèmes, nous avons mis en place un système de détection global qui nous permet de suivre l'état des réponses du serveur.

Si le serveur envoi :

- un **message** : La variable de suivi d'état passe à 1.
- une **seed** : La variable de suivi d'état passe à 0.
- une **information nulle** : La variable de suivi d'état passe à -1.

Ainsi, lorsque la variable de suivi d'état stocke la valeur 0, nous désactivons le chat et créons une salle de jeu en utilisant les informations des deux joueurs. À la fin d'une partie, le chat est réactivé et les joueurs ont la possibilité de discuter, de lancer une nouvelle partie ou de quitter le jeu.

Cette approche de détection globale nous a permis de résoudre les problèmes d'asynchronisme entre le chat et le jeu, en assurant une synchronisation plus fluide et une meilleure coordination entre les différentes parties du jeu. En intégrant ces modifications, nous avons amélioré l'expérience de jeu globale et offert aux joueurs une interface plus cohérente et conviviale.

2.3.6 Changement majeur au niveau du client

Nous avons apporté un changement majeur au niveau du client depuis nos deux premières soutenances. Initialement, nous utilisions deux clients distincts pour gérer les informations du chat et celles du mode multijoueur. Cependant, nous avons réalisé que cette approche n'était pas optimale. Nous avons donc pris la décision de fusionner le client du chat et celui du multijoueur en un seul client.

Cette fusion a entraîné plusieurs avantages significatifs. Tout d'abord, elle a simplifié la détection et le traitement des informations provenant du serveur. En ayant un seul client, nous avons pu mettre en place une logique de parsing plus efficace, ce qui nous a permis de mieux interpréter les données reçues et de les utiliser de manière appropriée.

De plus, cette fusion a grandement contribué à fluidifier la synchronisation entre le client et le serveur, comme nous l'avons mentionné précédemment. En ayant un seul client responsable à la fois du chat et du mode multijoueur, nous avons pu mieux coordonner les échanges d'informations et les mises à jour, ce qui a permis une expérience de jeu plus fluide et sans interruption.

En résumé, nous avons réalisé un changement majeur au niveau du client en fusionnant les clients du chat et du mode multijoueur en un seul client. Cette décision a simplifié la détection des informations via le parsing et a contribué à une meilleure synchronisation entre le client et le serveur, améliorant ainsi l'expérience de jeu globale.

2.4 Générateur de texte

2.4.1 Introduction

Dans ce projet, notre objectif est de créer des lignes de code pour que les joueurs puissent les taper. Pour y parvenir, nous avons développé un système qui peut générer un code à partir d'une base de données. L'une des tâches les plus importantes dans le traitement du langage est la détermination du sens d'un mot dans un contexte particulier.

Dans la suite de ce rapport, nous allons expliquer en détail notre approche et comment elle fonctionne, ainsi que les algorithmes que nous avons utilisés pour développer notre solution.

2.4.2 Commencement

Pour commencer, nous avons dû nous pencher sur la tâche complexe de créer une liaison entre un mot futur et un mot actuel. Nous avons exploré différentes approches, notamment les chaînes de Markov. Cependant, nous avons rapidement réalisé que la modélisation des probabilités en C pour obtenir l'état suivant était une tâche ardue. En effet, la modélisation des probabilités conditionnelles peut devenir très complexe, en particulier lorsque le nombre de variables augmente. Nous avons donc réfléchi et nous nous sommes dit que pour y arriver nous allons fonctionner comme cela :

- obtenir une string via un notre base de données
- créer une liste chaînée avec tous les mots comme valeur de la liste
- récupérer tous les index d'un mot recherché
- renvoyer le mot suivant grâce à aux indexs trouvés précédemment

2.4.3 Première partie

Pour ce qui est du programme en lui même, nous avons du mettre en place plusieurs fonctions. La toute première qui est utilisé dans le programme est la fonction char *filetochar qui permet d'obtenir une chaîne de caractère via un fichier. Par la suite nous avons donc implémenté la fonction struct list *chartolist qui comme énoncé précédemment sert à créer une list qui va prendre comme valeur tous les mots de la base de donnée. Cette fonction renvoie par exemple ceci :

```
ethandage@ethandage-Inspiron-5515:~/Documents/S4/Digit-Dash/game/find_word$ ./a.out "return"  
[ (int; i=0;; i<50;; i++); (; if; (i%2==0); (; printf("Win!");; break;;
```

FIGURE 6 – Résultat de la fonction

Ensuite, nous avons mis en place la fonction struct list *chartolist, qui est utilisée pour créer une liste chaînée contenant tous les mots de notre base de données.

Cette liste est ensuite utilisée pour déterminer le mot futur en utilisant la fonction struct inlist *create_index_list. Cette dernière fonction retourne une liste chaînée qui contient les index du mot donné. Cette liste nous est utile pour déterminer le mot futur en récupérant un de ces index de manière aléatoire, en lui ajoutant 1, et en récupérant ensuite le mot correspondant dans la liste du début (avec les mots).

Toutes nos fonctions nous donnait ce résultat :

```
ethandage@ethandage-Inspiron-5515:~/Documents/S4/Digit-Dash/game/find_word$ ./a.out "if"  
(i*y>i+y)  
ethandage@ethandage-Inspiron-5515:~/Documents/S4/Digit-Dash/game/find_word$ ./a.out "if"  
(i-y!=0)  
ethandage@ethandage-Inspiron-5515:~/Documents/S4/Digit-Dash/game/find_word$ ./a.out "if"  
(i-y!=0)  
ethandage@ethandage-Inspiron-5515:~/Documents/S4/Digit-Dash/game/find_word$ ./a.out "if"  
(i==0)
```

FIGURE 7 – Résultat de l'ensemble des fonctions

2.4.4 Seconde partie

En ce qui concerne l'avancement de cette partie, nous avons réalisé plusieurs actions pour améliorer notre base de données. Tout d'abord, nous avons augmenté la capacité de notre base de données en langage C pour mieux répondre aux besoins de nos utilisateurs.

De plus, nous avons introduit la création d'un code que l'utilisateur peut coder lui-même. Ce code a été élaboré en utilisant les fonctions décrites précédemment, qui ont été répétées pour renforcer les compétences en dactylographie. Cela permet aux utilisateurs de personnaliser leur expérience de dactylographie en créant leur propre code, adapté à leurs besoins spécifiques.

En résumé, ces mesures ont permis d'améliorer la qualité de notre plateforme de dactylographie et de répondre aux besoins variés des utilisateurs en matière de langages et de personnalisation de leur expérience de dactylographie.

```
● ethandage@ethandage-Inspiron-5515:~/Documents/S4/Digit-Dash/game/find_word$ ./markov
int len = (resultat + '0';
decimal /= 2;
}
if (n<2)
return 1;
}

void count_words(char *text, int len = (resultat + '0';
decimal /= 2;
}
if (n<2)
return 1;
}
```

FIGURE 8 – Exemple d'un résultat concluant

Néanmoins, il nous restait quand même certains problèmes à résoudre comme le montre la photo suivante. De temps en temps, une répétition dans le résultat de notre fonction avait lieu.

```
int chaine_vers_entier(char chaine[])
{
    int i + (binary[i] - '0');
}

int chaine_vers_entier(char chaine[])
{
    int i + (binary[i] - '0');
}
```

FIGURE 9 – Exemple d'un mauvais résultat

2.4.5 Depuis la dernière soutenance

Lors de notre dernière soutenance, nous avons identifié certains points négatifs qui nécessitaient des corrections. En particulier, nous avons remarqué une répétition assez fréquente lors de l'appel de notre fonction, ce qui engendrait des problèmes ultérieurs. Conscients de l'importance de résoudre cette problématique, nous avons décidé de repartir pratiquement de zéro.

Nous avons réalisé qu'en laissant cette répétition se perpétuer, cela aurait pu compromettre la qualité globale de notre projet. Ainsi, nous avons pris la décision de revoir notre approche et de repenser notre code de manière plus efficace et optimale.

Nous avons décidé de remplacer les listes chaînées par des dictionnaires. En effet il est plus pratique de les utiliser pour résoudre notre problème. Nous avons donc créé les structures suivantes :

```
typedef struct Node
{
    char *line;
    struct Node *next;
} Node;

typedef struct
{
    char *line;
    Node *next_lines;
    int count;
} DictEntry;
```

FIGURE 10 – Structures utilisées

Cette structure permet de créer un dictionnaire qui prend comme information chaque ligne de notre base de donnée et qui prend également un pointeur qui va être dirigé vers les probables futures lignes de code. En effet, sera stocké dans `next_lines` toutes les lignes apparaissant après notre ligne actuelle.

Nous avons donc été confrontés à la nécessité de tout reprendre à zéro et de repartir avec les nouvelles structures que nous venions de créer. Pour ce faire, nous avons mis en place les fonctions suivantes.

Tout d'abord, nous avons implémenté la fonction `generate_dict`, qui nous permet de générer un dictionnaire à partir de notre base de données et de la taille souhaitée pour ce dictionnaire. Cette fonction nous a permis de créer une structure de données organisée et adaptée à nos besoins. En exploitant les informations de la base de données, nous

avons pu construire un dictionnaire cohérent et riche.

Par la suite, nous avons logiquement mis en place la fonction de `print_dict` qui nous permet d'afficher le contenu de ce dictionnaire. Cette fonction était essentielle pour vérifier la conformité des données et s'assurer que notre dictionnaire était correctement construit.

Voici une exemple de notre dictionnaire :

```
double precision = 0.00001; -> while (fabs(resultat * resultat - nombre) > precision) {
while (fabs(resultat * resultat - nombre) > precision) { -> resultat = (resultat + nombre / resultat) / 2.0; }
resultat = (resultat + nombre / resultat) / 2.0; } -> return resultat; }
int factorial(int n) { -> if (n == 0 || n == 1) {
if (n == 0 || n == 1) { -> return 1;
return n * factorial(n - 1); } } -> void reverseString(char* str) {
void reverseString(char* str) { -> int length = 0;
int length = 0; -> while (str[length] != '\0') {while (str[length] != '\0') {
while (str[length] != '\0') { -> length++; }length++; }
length++; } -> int start = 0;int start = 0;
int start = 0; -> int end = length - 1;int end = length - 1;
int end = length - 1; -> while (start < end) {while (start < end) {
while (start < end) { -> if (str[start] != str[end]) {char temp = str[start];
char temp = str[start]; -> str[start] = str[end];
str[start] = str[end]; -> str[end] = temp;
str[end] = temp; -> start++;
str[end] = temp; -> start++;
}
```

FIGURE 11 – Exemple de notre dictionnaire

Grâce à ce dictionnaire bien établi, nous avons ensuite développé la fonction `generate_lines`. Cette fonction nous permet de générer un certain nombre de lignes de code en utilisant le dictionnaire préalablement créé. Cette fonction a été conçue pour produire des lignes de code variées et réalistes, en tirant profit des informations stockées dans le dictionnaire.

Pour garantir une génération fluide des lignes de code, nous avons également mis en place la fonction `get_next_line`. Cette fonction joue un rôle crucial en nous permettant de trouver la ligne suivante après celle que nous venons de créer. Grâce à la consultation du dictionnaire et à l'utilisation d'un processus aléatoire, nous avons pu maintenir une certaine diversité et un aspect dynamique dans la génération des lignes de code.

L'implémentation de ces différentes fonctions a été une étape importante dans notre processus de développement. Elles ont contribué à la construction d'un système solide et fonctionnel, nous permettant de générer des lignes de code de manière automatisée et cohérente, en tirant parti d'un dictionnaire bien structuré et d'une logique aléatoire bien maîtrisée.

Voici maintenant un exemple de notre fonction qui génère des lignes de code.

```
ethandage@ethandage-Inspiron-5515:~/Documents/S4/Digit-Dash/game/find_word$ ./markov
free(matrix); }
char* copyString(const char* str) {
    int length = strlen(str);
    char* reversed = (char*)malloc((length + 1) * sizeof(char));
    for (int i = 0; i < length; i++) {
ethandage@ethandage-Inspiron-5515:~/Documents/S4/Digit-Dash/game/find_word$ ./markov
i = 1;
int* createArray(int size) {
    int* array = (int*)malloc(size * sizeof(int));
    return array; }
int* resizeArray(int* array, int newSize) {
```

FIGURE 12 – Résultat final

2.5 Affichage du jeu

2.5.1 Introduction

L’affichage est un élément crucial pour tout jeu, y compris ceux joués sur le terminal de commande. Bien que les jeux en ligne de commande ne disposent pas des graphismes et des animations sophistiqués des jeux modernes, ils peuvent offrir une expérience de jeu immersive et engageante grâce à un affichage bien conçu. Un bon affichage peut rendre le jeu facile à lire, avec des couleurs et des contrastes bien choisis pour une meilleure lisibilité. Il peut également être utilisé pour afficher des informations importantes telles que le score, les niveaux, les vies restantes et les instructions de jeu.

De plus, l’affichage peut aider à créer une ambiance appropriée pour le jeu, en utilisant des couleurs et des motifs adaptés à l’univers du jeu. L’affichage est donc un élément essentiel de tout jeu, qu’il soit joué sur un ordinateur de bureau sophistiqué ou sur un terminal de commande minimaliste.

2.5.2 Code couleur

Lorsque le joueur lance une partie, le jeu commence par afficher sur le terminal le code à écrire. Ce code est affiché en gris clair, presque transparent, pour que le joueur puisse voir ce qu’il doit écrire tout en distinguant clairement le code original. Le code à écrire prend la forme d’un motif, qui peut être une série de lignes de code ou d’instructions. Le niveau approprié est chargé grâce au retour du générateur de texte expliqué précédemment, le programme écrit le code dans la console du joueur, ligne par ligne.

Cependant, l'indentation (la disposition du code en blocs) est ignorée dans cette version de développement, afin de faciliter les tests. Le joueur doit ensuite écrire le code demandé, caractère par caractère, et le programme vérifie chaque caractère saisi pour s'assurer que le joueur suit correctement le modèle affiché.

Les caractères correctement saisis sont affichés en vert pour indiquer leur validité, tandis que les caractères incorrects sont affichés en rouge. Si aucun caractère n'a été saisi, le programme affiche le caractère correspondant dans le modèle en gris. Le joueur doit ainsi saisir tout le code correctement pour terminer le niveau et passer au suivant.

2.5.3 Fonctionnement

Dans cette section, nous allons présenter le fonctionnement du jeu ainsi que détailler toutes les étapes du développement qui nous ont permis d'atteindre le résultat final, aboutissant à une expérience de jeu captivante et immersive.

2.5.3.1 Ancienne implémentation

Auparavant, pour afficher le niveau à l'utilisateur, le programme utilisait une méthode simple consistant à lire dans un fichier contenant les différents niveaux et à les afficher ligne par ligne sur le terminal. De plus, pour stocker les entrées de l'utilisateur et afficher les couleurs correspondantes, le programme stockait les entrées dans un fichier temporaire afin de garder en mémoire l'intégralité des inputs.

Cette méthode de stockage des entrées n'était pas très efficace et adaptée au fonctionnement du programme, car elle nécessitait une grande quantité de stockage de données et ralentissait l'exécution du jeu. De plus, elle rendait la gestion des entrées plus complexe, car le programme devait parcourir le fichier temporaire à chaque nouvel in-

put pour vérifier la validité des caractères saisis et mettre à jour les couleurs d’affichage.

Par conséquent, cette méthode a été remplacée pour simplifier la gestion des entrées et améliorer les performances du jeu.

2.5.3.2 Nouvelle implémentation

Dans la nouvelle version du jeu, le générateur de texte renvoie maintenant un tableau de chaînes de caractères. Cette amélioration a grandement simplifié l’affichage du jeu, car il suffit de parcourir le tableau et d’afficher une chaîne de caractères par ligne pour afficher le code à écrire.

En outre, le stockage des entrées de l’utilisateur a également été amélioré. Le même fonctionnement que pour le générateur de texte est utilisé, à savoir un tableau de chaînes de caractères alloué en mémoire de la même taille que celui renvoyé par le générateur de texte.

Chaque chaîne de caractères à l’intérieur du tableau est initialisée avec un caractère nul. Ainsi, à chaque caractère entré par l’utilisateur, le caractère nul est décalé d’une position pour laisser la place au nouveau caractère saisi.

Cette méthode permet de stocker efficacement les entrées de l’utilisateur tout en minimisant l’utilisation de la mémoire et en améliorant les performances du jeu.

2.5.4 Statistiques du joueur

Les statistiques occupent une place importante dans le jeu, car elles permettent au joueur de mesurer sa performance et de suivre sa progression au fil du temps. Ces statistiques sont affichées pendant la partie. Le but du jeu est de réaliser le meilleur score possible, que ce soit en comparaison avec un autre joueur ou même par rapport à son propre meilleur score.

Pour cela, plusieurs statistiques sont disponibles, telles que le nombre de victoires, la fréquence de frappe ou même le taux d'entrées correctes. Ces statistiques fournissent des informations clés pour aider les joueurs à comprendre leur niveau de compétence et à identifier les domaines où ils peuvent améliorer leur performance.

De plus, elles peuvent également être utilisées pour suivre les progrès des joueurs et leur permettre de fixer des objectifs pour atteindre des scores plus élevés.

2.6 TUI

2.6.1 Pourquoi ncurses

Lors du développement de notre jeu dans le terminal, nous avons choisi d'utiliser la librairie ncurses, et nous sommes convaincus que c'était la meilleure option. Ncurses offre des fonctionnalités avancées pour contrôler l'interface utilisateur et gérer les événements, ce qui était essentiel pour créer une expérience de jeu immersive.

Grâce à ncurses, nous avons pu créer une interface utilisateur interactive en utilisant des fenêtres, des panneaux et des boîtes de dialogue. Cela nous a permis d'organiser et d'afficher les différents éléments du jeu de manière conviviale, en segmentant l'écran du terminal. L'inter-

face était claire et facile à comprendre pour les joueurs.

La gestion des événements d'entrée tels que les touches du clavier était également un aspect crucial de notre jeu. Ncurses a simplifié cette tâche en offrant des fonctionnalités pour détecter les touches pressées les joueurs. Nous avons ainsi pu créer des interactions fluides entre vous et le jeu, ce qui a rendu l'expérience plus immersive et engageante.

En résumé, l'utilisation de la librairie ncurses pour notre jeu dans le terminal a été un choix judicieux. Ses fonctionnalités avancées de contrôle de l'interface utilisateur et de gestion des événements ont permis de créer une interface conviviale et d'offrir une expérience de jeu immersive. Nous sommes satisfaits des résultats obtenus grâce à l'utilisation de cette librairie.

2.6.2 Menu

Nous avons créé un menu pour notre jeu en utilisant la librairie ncurses. Le menu est composé d'un titre du jeu affiché en ASCII-art, ce qui ajoute une touche visuelle attrayante. Les caractères ASCII-art sont utilisés pour représenter graphiquement le titre du jeu avec des caractères et des symboles spécifiques. Cela donne au menu un aspect unique et distinctif.

En plus du titre, nous avons ajouté quatre boutons pour les différentes options du jeu : "Entraînement", "Multijoueur", "Aide" et "Quitter". Ces boutons sont affichés de manière interactive, ce qui signifie que les joueurs peuvent les sélectionner en utilisant les touches du clavier. Lorsqu'un bouton est sélectionné, son apparence peut changer pour indiquer visuellement qu'il est activé.

Pour mettre en valeur le menu, nous l'avons entouré d'un cadre de deux couleurs. Cela crée une délimitation claire entre le menu et le reste de l'interface. Le cadre peut être personnalisé en termes de couleurs et d'épaisseur pour correspondre à l'esthétique globale du jeu.

2.6.3 Jeu

Nous avons développé un affichage dynamique et interactif pour notre jeu. Cet affichage est composé de plusieurs cadres, chacun servant à afficher des éléments spécifiques.

En haut de l'écran, nous avons placé un cadre qui affiche une barre de progression du jeu. Cette barre de progression permet aux joueurs de visualiser visuellement l'avancement de leur partie. Elle se met à jour en temps réel, offrant ainsi une indication claire de la progression et de l'état actuel du jeu.

Juste en dessous, nous avons ajouté un autre cadre qui contient le code que le joueur doit écrire. Ce code s'affiche au fur et à mesure de la progression du jeu. Les joueurs doivent interagir avec ce cadre en écrivant le code approprié pour progresser dans le jeu. Grâce à ncurses, nous avons pu rendre ce cadre interactif, permettant aux joueurs de saisir leur code directement à partir du clavier.

En haut à droite de l'écran, nous avons inclus le nom du jeu en petit ASCII art. Cela ajoute une touche visuelle supplémentaire à l'interface et permet de personnaliser l'affichage pour qu'il corresponde à l'esthétique globale du jeu. Cette petite touche artistique ajoute une identité visuelle unique à notre jeu.

En dessous du nom du jeu, nous avons placé un cadre qui affiche les statistiques du joueur en temps réel. Ces statistiques peuvent inclure des informations telles que le score, le temps écoulé, le nombre d'ennemis vaincus, etc. Grâce à la flexibilité de ncurses, nous avons pu actualiser ces statistiques en temps réel pour refléter les actions du joueur pendant le jeu.

2.6.4 Aide

Lorsque les joueurs ont besoin d'aide ou souhaitent comprendre le fonctionnement du jeu, nous avons créé un affichage dédié en utilisant la librairie ncurses. Cet affichage est conçu de manière simple mais efficace pour fournir aux joueurs les règles du jeu, ainsi que des astuces et des conseils sur le jeu et les commandes.

Lorsque le joueur sélectionne le bouton "Aide", une nouvelle fenêtre s'ouvre à l'écran, présentant les règles du jeu de manière claire et concise. Les règles essentielles sont affichées, expliquant les objectifs du jeu, les mécanismes de jeu et tout autre élément pertinent pour comprendre le fonctionnement global du jeu.

En plus des règles, nous avons inclus différents conseils et astuces pour aider les joueurs à progresser dans le jeu. Ces astuces peuvent être des recommandations sur les meilleures stratégies à adopter, des informations sur les éléments à collecter ou des conseils pour surmonter des défis spécifiques. Cela permet aux joueurs d'améliorer leurs compétences et de découvrir des aspects du jeu qu'ils pourraient ne pas avoir explorés auparavant.

De plus, nous avons également fourni des informations sur les commandes du jeu dans cette fenêtre d'aide. Les joueurs peuvent trouver une liste des commandes clés et leur fonctionnement associé. Cela les aide à mieux comprendre comment interagir avec le jeu et à utiliser efficacement les différentes fonctionnalités mises à leur disposition.

2.6.5 Quitter

Le bouton "Quitter" dans notre jeu offre une fonctionnalité simple mais essentielle. Lorsque les joueurs sélectionnent ce bouton, toutes les fenêtres créées par ncurses se ferment et le jeu se termine. Cela permet aux joueurs de quitter le jeu de manière propre et ordonnée, en s'assurant que toutes les ressources utilisées par ncurses sont libérées correctement.

Fermer correctement les fenêtres créées par ncurses est crucial pour éviter les problèmes de mémoire et garantir une exécution fluide du jeu. En libérant les ressources de manière appropriée, nous nous assurons que le jeu se termine sans laisser de résidus ou d'erreurs.

2.7 Chat

2.7.1 Première soutenance

Lors de notre première soutenance, nous avons réussi à accomplir plusieurs étapes importantes du développement de notre programme. Nous avons notamment mis en place la fonctionnalité de connexion au serveur à l'aide de sockets, permettant ainsi d'établir une communication bidirectionnelle entre le client et le serveur.

Une fois connecté, notre programme demande à l'utilisateur de saisir son nom d'utilisateur, afin de l'identifier lors des échanges ultérieurs. Cette étape a été intégrée avec succès, permettant ainsi une personnalisation de l'expérience utilisateur.

Par la suite, nous avons implémenté la fonctionnalité permettant à l'utilisateur d'envoyer des messages au serveur. L'utilisateur est invité à saisir le message qu'il souhaite transmettre, et notre programme l'encapsule dans une requête sous la forme d'une chaîne de caractères, de la forme "Nom : message". Cette approche permet une structuration claire des données échangées.

Pour confirmer la réception du message, le serveur renvoie une réponse indiquant "Received". Nous avons intégré cette fonctionnalité dans notre programme, offrant ainsi un retour immédiat à l'utilisateur.

L'ensemble de ces fonctionnalités a été intégré dans une boucle While, permettant ainsi à l'utilisateur d'envoyer autant de messages qu'il le souhaite au serveur. Cette boucle offre une interaction fluide et réactive, en garantissant une communication continue entre le client et le serveur.

Au terme de cette première soutenance, nous sommes parvenus à mettre en place une base solide pour notre programme, permettant une connexion réussie au serveur, l'envoi de messages personnalisés et une confirmation de réception. Nous sommes satisfaits des résultats obtenus et nous nous préparons à relever de nouveaux défis lors de la prochaine étape de développement.

2.7.2 Deuxième soutenance

Lors de notre deuxième soutenance, nous avons réalisé des avancées significatives dans le développement de notre programme de chat en ligne, permettant une communication de type "user-to-user" (U2U) entre deux utilisateurs.

Nous avons mis en place l'utilisation des sockets pour assurer la communication bidirectionnelle entre les deux utilisateurs. De plus, nous avons intégré la bibliothèque de threads "pthread.h" afin de gérer les tâches simultanées et garantir une expérience de chat fluide.

La fonction principale de notre programme, appelée "u2u", a été développée pour établir une connexion avec le serveur en utilisant une adresse IP et un port spécifiques. Elle crée ensuite deux threads distincts pour gérer les messages entrants et sortants.

La première fonction, "sendmessage", est responsable de l'envoi des messages entre les utilisateurs. Elle demande à l'utilisateur de saisir son nom, puis utilise une boucle while pour lire les messages entrés par l'utilisateur depuis la console. Les messages sont ensuite stockés dans une structure de données appelée "Chatinfo" et envoyés au serveur via le socket.

La deuxième fonction, "receivemessage", est utilisée pour recevoir les messages envoyés par l'autre utilisateur. Elle utilise également une boucle while pour lire les messages reçus depuis le socket et les affiche sur la console de l'utilisateur.

Enfin, la fonction "u2u" crée les deux threads nécessaires pour gérer les messages entrants et sortants. Ces threads sont créés en utilisant la fonction "pthreadcreate" et sont joints à la fin de la fonction "u2u" à l'aide de "pthreadjoin". Ainsi, la fonction "u2u" se termine lorsque les threads sont correctement joints.

2.7.3 Objectif soutenance finale

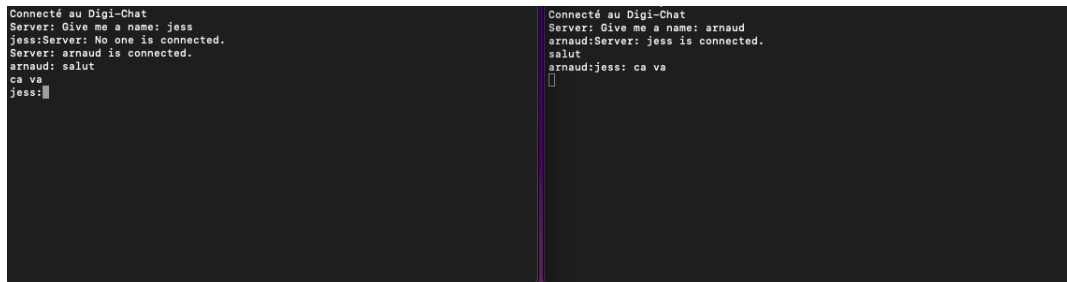
Pour la soutenance finale, notre objectif est de régler plusieurs problèmes dans notre programme de chat en ligne et d'ajouter des fonctionnalités supplémentaires pour améliorer l'expérience utilisateur.

Tout d'abord, nous allons résoudre le problème d'effacement du nom lors de la réception d'un message. Actuellement, le nom de l'utilisateur disparaît lorsque nous recevons un message, ce qui rend la lecture des conversations difficile. Nous allons corriger ce problème pour que le nom reste affiché en permanence.

Un autre problème lié est l'effacement des messages en cours de rédaction lors de la réception d'un nouveau message. Nous allons mettre en place une solution pour éviter cette perte accidentelle de messages non encore envoyés.

Ensuite, nous allons ajouter la bibliothèque NCURSES à notre programme. Cela nous permettra d'ajouter une interface utilisateur graphique attrayante. Nous créerons des fenêtres et des boîtes de dialogue pour afficher les messages de manière claire et organisée, améliorant ainsi la lisibilité et l'esthétique du chat.

En résumé, pour la soutenance finale, notre objectif est de régler les problèmes d'effacement du nom et des messages en cours de rédaction lors de la réception de nouveaux messages. Nous ajouterons également la librairie NCURSES pour créer une interface utilisateur graphique agréable dans le TUI.



```
Connecté au Digi-Chat
Server: Give me a name: jess
jess:Server: No one is connected.
Server: arnaud is connected.
arnaud: salut
ca va
jess:

Connecté au Digi-Chat
Server: Give me a name: arnaud
arnaud:Server: jess is connected.
salut
arnaud:jess: ca va
```

FIGURE 13 – Exposition des différents problèmes

2.7.4 Introduction des pthreads mutex

Dans cette partie de notre programme de chat en ligne, nous avons introduit l'utilisation de pthreads mutex. Les pthreads mutex (ou mutexes) sont des mécanismes de synchronisation utilisés pour assurer l'exclusion mutuelle lors de l'accès simultané à des ressources partagées par plusieurs threads.

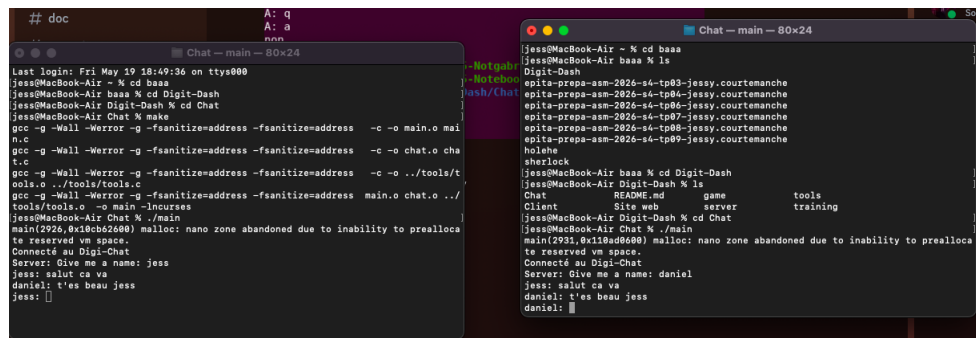
Dans notre nouveau code, nous avons ajouté la déclaration et l'initialisation du mutex pthreadmutex_t lock. Ce mutex sera utilisé pour protéger les sections critiques du code où l'accès à des variables partagées doit être contrôlé afin d'éviter des comportements indésirables.

Le programme comporte également deux threads : readfromserver et writetoserver. Le thread readfromserver est responsable de la lecture des messages du serveur, tandis que le thread writetoserver est chargé d'envoyer les messages au serveur.

Lors de l'exécution du thread readfromserver, nous utilisons le mutex pour protéger la section critique où les messages reçus sont affichés dans la fenêtre de messages. Le mutex garantit qu'un seul thread à la fois peut accéder à cette section critique, évitant ainsi les problèmes de concurrence.

De même, lors de l'exécution du thread writetoserver, le mutex est utilisé pour protéger la section critique où les messages saisis par l'utilisateur sont affichés dans la fenêtre de messages. Encore une fois, le mutex assure l'exclusion mutuelle et garantit un accès sûr et synchronisé aux variables partagées.

L'ajout des pthreads mutex dans notre nouveau code renforce la cohérence et la fiabilité de la communication entre les threads et améliore la gestion des ressources partagées. Cela permet une exécution sans heurts des threads et prévient les conflits lors de l'accès aux données partagées.



```

# doc
A: q
A: a
nnn

Last login: Fri May 19 18:49:36 on ttys000
jess@MacBook-Air ~ % cd baas
jess@MacBook-Air baas % cd Digit-Dash
jess@MacBook-Air Digit-Dash % cd Chat
jess@MacBook-Air Chat % make
gcc -g -Wall -Werror -g -fsanitize=address -fsanitize=address -c -o main.o mai
n.c
gcc -g -Wall -Werror -g -fsanitize=address -fsanitize=address -c -o chat.o cha
t.c
gcc -g -Wall -Werror -g -fsanitize=address -fsanitize=address -c -o ../tools/t
ools.o ../tools/tools.c
gcc -g -Wall -Werror -g -fsanitize=address -fsanitize=address main.o chat.o ../
tools/tools.o -o main -lcurses
jess@MacBook-Air Chat % ./main
main(2926,0x18c02600) malloc: nano zone abandoned due to inability to prealloca
te reserved vm space.
Connecté au Digi-Chat
Server: Give me a name: jess
jess: salut ca va
daniel: t'es beau jess
jess:

jess@MacBook-Air ~ % cd baas
jess@MacBook-Air baas % ls
Digit-Dash
epita-prepa-asm-2026-s4-tp03-jessy.courtemanche
epita-prepa-asm-2026-s4-tp04-jessy.courtemanche
epita-prepa-asm-2026-s4-tp06-jessy.courtemanche
epita-prepa-asm-2026-s4-tp07-jessy.courtemanche
epita-prepa-asm-2026-s4-tp08-jessy.courtemanche
epita-prepa-asm-2026-s4-tp09-jessy.courtemanche
holehe
shezlock
jess@MacBook-Air baas % cd Digit-Dash
jess@MacBook-Air Digit-Dash % ls
Chat README.md game tools
Client Site web server training
jess@MacBook-Air Digit-Dash % cd Chat
jess@MacBook-Air Chat % ./main
main(2931,0x18c02600) malloc: nano zone abandoned due to prealloca
te reserved vm space.
Connecté au Digi-Chat
Server: Give me a name: daniel
jess: salut ca va
daniel: t'es beau jess
daniel:
  
```

FIGURE 14 – Noms qui ne se chevauchent pas grâce aux mutex

2.7.5 Introduction de Ncurses

Dans le cadre du développement de notre programme de chat en ligne, nous avons ajouté des fonctionnalités avancées en utilisant la bibliothèque NCURSES. Ces ajouts ont été intégrés dans le code que nous avons créé précédemment et ont permis de résoudre les problèmes mentionnés dans les objectifs.

L'une des améliorations clés a été l'utilisation de fenêtres distinctes pour gérer les messages et l'entrée utilisateur. Nous avons créé deux fenêtres : la fenêtre des messages (messagewin) et la fenêtre d'entrée des messages (inputwin). Ces fenêtres sont positionnées l'une au-dessus de l'autre pour permettre une visualisation claire et sans chevauchement des informations.

Grâce à cette organisation en fenêtres, nous avons résolu le problème d'effacement des messages non envoyés lors de la réception de nouveaux messages. En maintenant les messages en cours de rédaction dans la fenêtre d'entrée des messages (inputwin), ils restent visibles même lorsque de nouveaux messages sont reçus dans la fenêtre des messages (messagewin). Cela garantit que les messages en cours de composition ne sont pas perdus et permet à l'utilisateur de revoir et de corriger ses messages avant de les envoyer.

De plus, grâce à l'utilisation des fenêtres distinctes, nous avons pu prévenir le chevauchement des noms d'utilisateurs lors de la réception de messages. En affichant les noms d'utilisateurs dans la fenêtre d'entrée des messages (inputwin), ils restent séparés des messages reçus dans la fenêtre des messages (messagewin). Cela garantit que les noms d'utilisateurs ne se superposent pas et permet une meilleure lisibilité des conversations.

En parallèle, l'utilisation de la bibliothèque NCURSES nous a également permis de gérer les couleurs de manière avancée. Nous avons créé des paires de couleurs personnalisées à l'aide de `initpair()` pour attribuer des couleurs distinctes aux différents éléments de l'interface. Par exemple, nous avons utilisé la paire de couleurs 1 pour afficher les messages de l'utilisateur en vert et la paire de couleurs 2 pour afficher les messages reçus en rouge. Nous avons également utilisé la paire de couleurs 4 pour afficher les messages du serveur en cyan.

Cette gestion avancée des couleurs grâce à NCURSES a amélioré la lisibilité et l'esthétique globale de l'interface utilisateur. Les couleurs distinctes pour les différents types de messages facilitent la distinction entre les messages de l'utilisateur, les messages reçus et les messages du serveur, offrant ainsi une meilleure compréhension des conversations.

En résumé, l'ajout des fenêtres distinctes et l'utilisation avancée de la bibliothèque NCURSES ont permis de résoudre les problèmes mentionnés dans les objectifs. Les fenêtres séparées pour les messages et l'entrée utilisateur ont empêché l'effacement des messages non envoyés et le chevauchement des noms d'utilisateurs. De plus, la gestion avancée des couleurs a amélioré la lisibilité et l'esthétique de l'interface utilisateur.

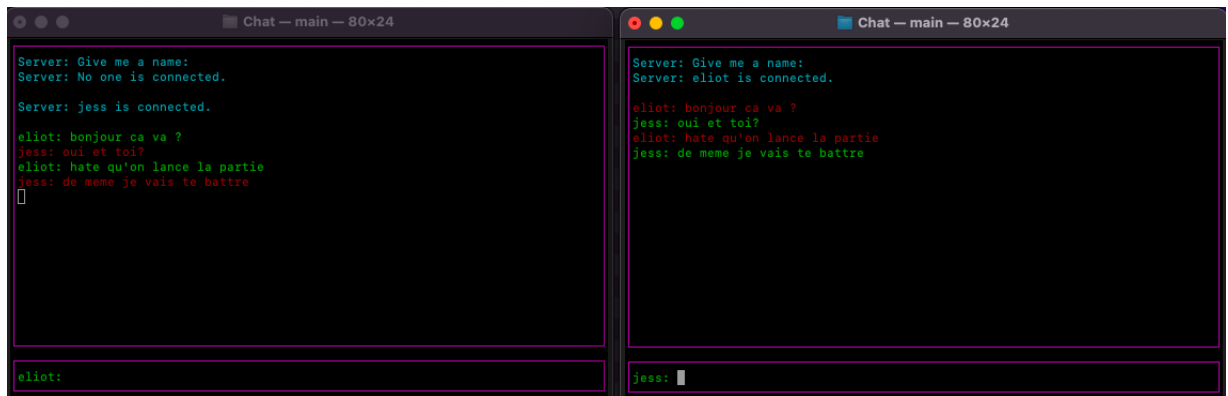


FIGURE 15 – Nouveau chat grace à l’ajout de Ncurses

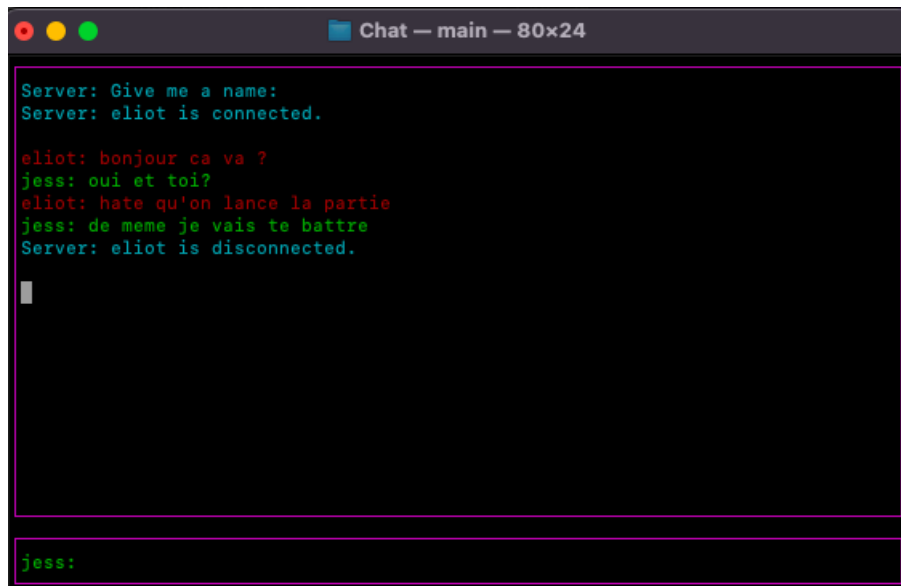


FIGURE 16 – Message du serveur

2.8 Site web

Dans le cadre du développement de notre site web, nous avons ajouté deux sous-pages supplémentaires : Chat et Gameplay. Ces sous-pages sont conçues pour présenter et mettre en valeur les fonctionnalités clés de notre jeu.

La sous-page Chat offre un aperçu de la fonctionnalité de chat en ligne que nous avons développée. Nous expliquons comment les utilisateurs peuvent interagir les uns avec les autres en temps réel, discuter, échanger des messages et créer des conversations privées ou de groupe. Nous mettons en avant la convivialité et l'accessibilité de notre système de chat, permettant aux joueurs de rester connectés et de partager leur expérience de jeu de manière interactive.

La sous-page Gameplay, quant à elle, est dédiée à la présentation des mécanismes et des fonctionnalités de jeu de notre programme. Nous expliquons les objectifs du jeu, les règles, les commandes et les différentes étapes ou niveaux que les joueurs peuvent rencontrer. Nous mettons en avant les éléments clés de gameplay qui rendent notre jeu unique et engageant. À travers des captures d'écran, des vidéos ou des animations, nous offrons aux visiteurs du site un aperçu visuel des expériences de jeu qu'ils peuvent attendre de notre programme. Nous veillons à présenter les aspects les plus captivants et excitants du gameplay pour susciter l'intérêt et l'engagement des joueurs potentiels.

Elles permettent aux visiteurs de découvrir les fonctionnalités clés du jeu, d'avoir un aperçu de l'expérience utilisateur et de susciter leur intérêt pour le télécharger ou l'essayer par la suite.

En résumé, les sous-pages Chat et Gameplay de notre site web offrent un aperçu des fonctionnalités de chat en ligne et du gameplay de notre jeu. Elles jouent un rôle essentiel dans la présentation de notre jeu aux visiteurs et les incitent à explorer davantage l'expérience de jeu offerte.

2.9 Conclusion

Nous avons donc achevé notre projet, un projet qui nous était cher à tous les membres de l'équipe. Chacun d'entre nous a dû faire face à plusieurs difficultés lors de sa réalisation. Cependant, nous avons réussi à les surmonter avec détermination et persévérance.

Malgré les obstacles rencontrés, nous sommes fiers d'avoir mené ce projet à terme. Il représente une réalisation significative pour nous tous, tant sur le plan personnel que professionnel. Chaque membre a contribué de manière unique à sa réussite, apportant son expertise et son engagement.

Bien que le projet soit désormais terminé, nous sommes conscients que sa trajectoire ne s'arrête pas là. Il pourrait continuer à évoluer et à s'améliorer avec le temps.

En somme, la réalisation de ce projet représente un accomplissement pour nous tous. Nous avons surmonté les difficultés avec détermination et nous envisageons avec optimisme son potentiel d'amélioration continue à l'avenir.