# PROJET MCS
# Création d’un mastermind multijoueur

## PARTIE 1 : Spécification

### 1.1 Explication du projet

Dans le cadre de ce projet, nous visons à développer une version modifiée du jeu de Mastermind. Le jeu pourra être en version multijoueur. Il permet à 1 à 4 joueurs de se connecter à un même serveur pour jouer ensemble. Une fois que tous les joueurs sont connectés aux serveurs et qu'ils sont prêts, le jeu démarre et chaque joueur tente de deviner la combinaison secrète en un maximum de 12 tentatives. L'objectif sera de trouver la combinaison de couleurs secrète avec le moins de tentatives possible et en moins de temps possible.

### 1.2 Fonctionnalités

#### Création de la partie :

Les joueurs doivent se connecter au serveur, lorsqu’ils cochent qu’ils sont prêts, la partie démarre. Lorsqu’un seul joueur est connecté au serveur, il peut faire une partie seul. Lorsque plusieurs joueurs sont connectés, il faut attendre que tous les joueurs mentionnent qu’ils soient prêts pour démarrer la partie. Les joueurs auront la possibilité de rejoindre une partie existante en entrant l'adresse IP du serveur sur laquelle ils souhaitent se connecter.

#### Interface de jeu :

La partie commence lors de la compilation du client et du serveur. Par le terminal, il est demandé au client de rentrer l’IP et le port du serveur auquel il souhaite se connecter. Une fois connecté au serveur, le client envoie au serveur qu’il est “ready”, c'est-à-dire qu’il est prêt à commencer la partie. L’interface signale ensuite au client qu’il attend la réponse des autres joueurs connectés au serveur avant de démarrer la partie. L’interface informe le nombre de joueurs dans la partie et l’identifiant de celui-ci.

Une fois la partie lancée, l’interface utilisateur est créée sur le terminal de la façon suivante :


Figure 1 : IHM au début de la partie

Les 12 lignes de 4 carrés représentent l'espace pour chaque tentative du joueur. Lorsque le joueur propose une combinaison de quatres caractères, il est mémorisé et inséré sur la première ligne. Les deux colonnes à côté représentent les indices pour le joueur : la première indique le nombre de couleurs correctes et **bien** placées, présentes dans la combinaison proposée par le joueur. La seconde indique le nombre de couleurs correctes et **mal** placées de la combinaison.

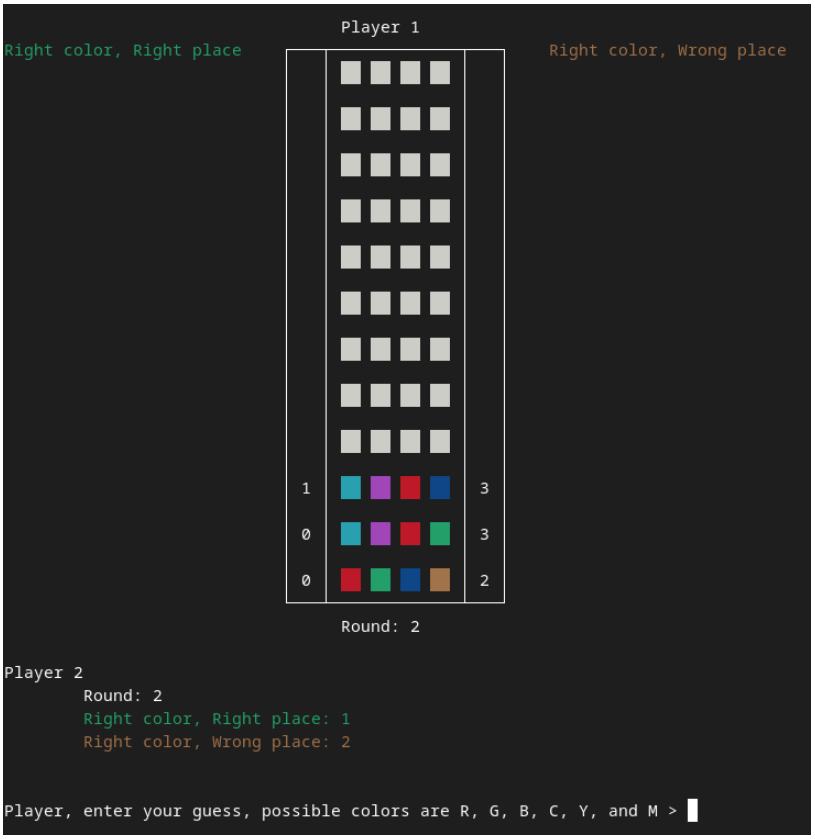


Figure 2 : IHM durant la partie

Le tableau en bas du terminal montre les performances des autres joueurs actualisées à chaque tour, avec leur nombre de tentatives effectuées, le nombre de couleurs trouvées, et le nombre de couleurs situées à la bonne position. Ainsi, le joueur peut déterminer son positionnement par rapport aux autres joueurs et accélérer ou non son jeu. Le but étant de trouver la combinaison le plus rapidement possible.

Les joueurs interagissent avec le jeu en utilisant des commandes simples basées sur l'entrée en majuscule des premières lettres de chaque couleur en anglais. Par exemple, pour proposer une combinaison de couleurs rouge, bleu, jaune et vert, le joueur entre simplement "RBYG". Le jeu reconnaît automatiquement ces commandes et les traite en conséquence.

### Client :

Le client est chargé de l'IHM, et ainsi de la gestion des entrées et sorties de l'utilisateur. Il vérifie aussi que la combinaison proposée par l'utilisateur est valide avant de l'envoyer au serveur. À la fin de la partie, le client se ferme automatiquement, on peut donc relancer le programme pour jouer une autre partie.

### Serveur :

Le serveur quant à lui va prendre en charge la synchronisation avec tous les clients, au début de la partie, il génère le code secret et attend que les clients lui communique les

coups choisis par l'utilisateur. Après traitement des coups choisi par l'utilisateur, il renvoie les résultats aux clients. À la fin de la partie, le serveur envoie aux clients le vainqueur et se réinitialise en vue de démarrer une nouvelle partie.

Gestion des parties :

Le serveur gère la création, la gestion et la clôture des parties une fois terminées. Le programme aura l'implémentation des règles du jeu du Mastermind, y compris la génération aléatoire de la combinaison secrète et la validation des coups des joueurs.

La partie se termine soit lorsqu'un joueur a trouvé la combinaison secrète, soit lorsque tous les joueurs ont atteint le maximum de 12 tentatives. A la fin de la partie, le numéro du gagnant est affiché ainsi que la combinaison secrète.

# PARTIE 2 : Conception

## 2.1 Technologies Utilisées

### Langage de programmation :

Le jeu est développé en utilisant le langage C, langage de programmation adapté à la manipulation des sockets et à la gestion de la logique de jeu.

### Sockets Stream INET :

Utilisation de sockets de type Stream INET pour la communication réseau entre les clients et le serveur.

### Interface Utilisateur :

L'interface utilisateur est directement sur le terminal.

### Architecture Client-serveur :

Le jeu suit une architecture client-serveur, où un serveur centralisé gère la logique du jeu et la communication entre les joueurs. Le client est uniquement chargé de l'IHM.

## 2.2 Architecture applicative en couches

Voici un tableau des différents protocoles utilisés pour chaque couche applicative nécessaires au fonctionnement du projet :

| Couche | Protocole | Description |
|---|---|---|
| Couche Présentation | - | Cette couche gère l'interface utilisateur côté client, affichant les informations du jeu et permettant les interactions avec le joueur. |
| Couche Application | - | La couche applicative est responsable de la logique du jeu, incluant la génération de la combinaison secrète, la vérification des propositions des joueurs et la gestion des tours. |
| Couche Session | Stream INET | La couche de session gère la communication entre le client et le serveur à travers des sockets de type Stream INET. |
| Couche Transport | TCP | Le protocole TCP est utilisé pour assurer la fiabilité et l'ordonnancement des paquets envoyés entre le client et le serveur. |
| Couche Réseau | IPv4 | Cette couche gère l'adressage et le routage des données entre les différents nœuds du réseau, en utilisant le protocole IPv4. |
| Couche Liaison | Ethernet | La couche de liaison assure la transmission des données au sein d'un réseau local, utilisant le protocole Ethernet. |

Figure 3 : Tableau d'architecture applicative en couches

## 2.3 Protocole applicatif : diagramme de séquences des requêtes/réponses

Dans ce protocole, tous les envoies de donnée sont suivi d'un acquittement avec un code de validation spécifique à chaque envoie, cela permet de garantir une bonne synchronisation client-serveur.

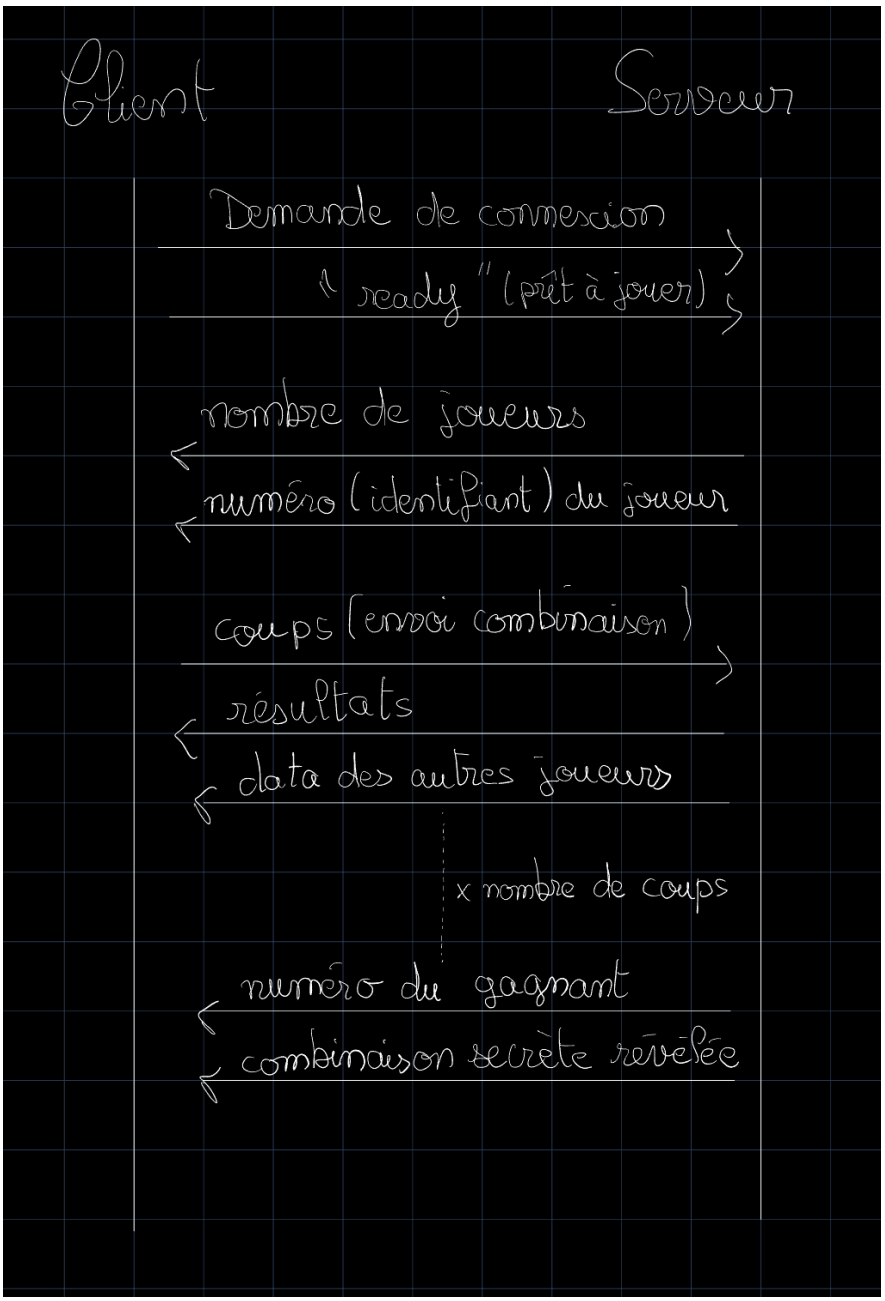


Figure 4 : Diagramme de séquence, requêtes/réponses

## 2.4 Scénarios d’utilisation

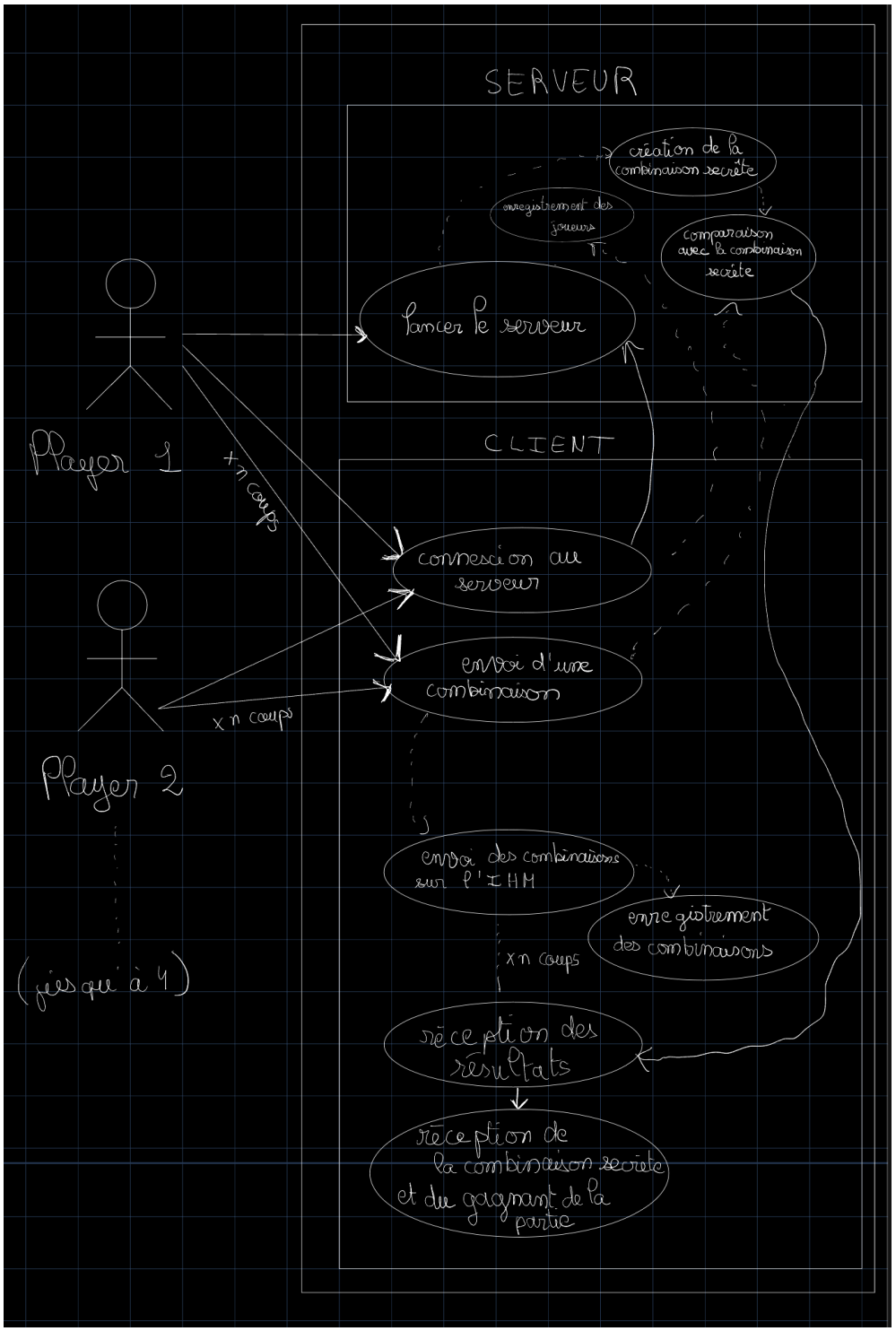


Figure 5 : diagramme des cas d’utilisation

## 2.5 Description fonctionnelle des traitements des requêtes

1) Traitement de la connexion des clients au serveur :

- Le serveur écoute les demandes de connexion entrantes des clients.

- Lorsqu'un client se connecte, le serveur lui attribue un identifiant unique pour la session.
- Le serveur envoie le nombre de joueur et son numéro d'identifiant (numéro de joueur)
- Le serveur crée la combinaison secrète et la garde en mémoire tout au long de la partie.

2) Analyse des propositions de combinaison des joueurs :

- Le serveur reçoit les propositions de combinaison des joueurs.
- Il analyse les combinaisons proposées et les compare à la combinaison secrète : Le serveur calcule le nombre de bonne couleur et de bonne position dans la combinaison pour déterminer les indices à renvoyer aux joueurs.
- Le serveur arrête la partie d'un joueur au bout de 12 tentatives.

3) Mise à jour de l'interface utilisateur des joueurs avec les informations reçues du serveur :

- Après chaque action, le serveur envoie des mises à jour aux clients. Les mises à jour incluent les résultats des autres joueurs.

# PARTIE 3 : Implémentation

## 3.1 Structure du code

Nous avons séparé le code du client et du serveur en deux dossiers distincts. Dans chaque dossier les sources et les includes sont contenus dans les dossiers src et include, ce qui permet une séparation claire des fichiers. Nous avons également créé les librairies libSocket et libUtils, afin de simplifier les communications avec les sockets et répertorier les fonctions non spécifiques au client et au serveur.

## 3.2 Analyse et explication du code

### Client :

Le code client est divisé en plusieurs fichiers, chacun responsable d'une partie spécifique de l'application. Voici une brève analyse de chaque fichier :

client.c :

- Ce fichier contient la fonction main() qui représente le point d'entrée du programme client.
- La fonction main() gère le déroulement principal du jeu, y compris l'affichage du menu, l'initialisation du jeu, la connexion au serveur et l'entrée dans la boucle principale du jeu.

- La boucle principale du jeu envoie la combinaison du joueur au serveur, reçoit le résultat de la combinaison, met à jour les données des autres joueurs et affiche l'état du jeu.
- Le jeu continue jusqu'à ce qu'il soit terminé, puis la fonction endGame() est appelée pour terminer le jeu.

clientCommunication.c :
- Ce fichier contient les fonctions responsables de la communication client-serveur.
- La fonction connexionWithServer() établit une connexion avec le serveur, permettant au joueur de spécifier l'adresse IP et le port du serveur, puis de se connecter en envoyant le message "ready".
- Les autres fonctions dans ce fichier gèrent l'envoi de la combinaison du joueur au serveur, la réception du résultat de la combinaison et la récupération des données des autres joueurs depuis le serveur.

clientInit.c :
- Ce fichier contient les fonctions d'initialisation du jeu client.
- Les fonctions initBoard(), initResult(), et initOtherPlayers() initialisent respectivement le plateau de jeu, le tableau des résultats et le tableau des autres joueurs.

clientShow.c :
- Ce fichier contient les fonctions de présentation du jeu client.
- La fonction showMenu() affiche le menu du jeu au début.
- La fonction showGame() affiche l'état actuel du jeu, y compris le plateau de jeu, les résultats des rounds précédents, et les données des autres joueurs.

Analyse et explication de la communication client-serveur :

La communication client-serveur est gérée principalement par le fichier clientCommunication.c. Voici un aperçu de cette partie du code :

connexionWithServer() :
- Cette fonction permet au joueur de se connecter au serveur en spécifiant l'adresse IP et le port.
- Une fois connecté, le joueur envoie le message "ready" pour indiquer qu'il est prêt à jouer.
- Le serveur répond en envoyant le nombre de joueurs dans la partie et l'index du joueur actuel.

sendCombination() :
- Cette fonction permet au joueur d'envoyer sa combinaison de couleurs au serveur.
- La combinaison est saisie par le joueur, validée et envoyée au serveur.

getResult() :
- Cette fonction récupère le résultat de la combinaison du joueur depuis le serveur.
- Le résultat est stocké dans le tableau des résultats du jeu.

fetchOtherClientsData() :

- Cette fonction récupère les données des autres joueurs depuis le serveur.
- Les données comprennent le nombre de bonnes couleurs à la bonne place et le nombre de bonnes couleurs à la mauvaise place.

## Serveur :

Le code serveur est structuré en plusieurs fichiers, chacun étant responsable d'une partie spécifique de la logique du jeu. Voici une analyse succincte de chaque fichier :

server.c :

- Ce fichier contient la fonction main() qui représente le point d'entrée du programme serveur. La fonction main() initialise les données du jeu et entre dans la boucle principale du jeu. La boucle principale du jeu gère les différentes étapes du jeu, telles que l'enregistrement des clients, la création du code secret, le démarrage du jeu et la fin du jeu.

serverInit.c :

- Ce fichier définit les fonctions pour initialiser les données du jeu et des joueurs. La fonction serverInit() initialise les données du jeu, y compris la liste des joueurs, le code secret, etc. La fonction _playerInit() initialise les données de chaque joueur, telles que son plateau de jeu et ses résultats.

serverCommunication.c :

- Ce fichier contient les fonctions pour gérer la communication entre le serveur et les clients. La fonction clientRegistration() gère l'enregistrement des clients en attendant leur connexion et en créant un nouveau thread pour chaque joueur. Les fonctions getPlayerChoice() et sendResult() sont responsables de la communication des choix des joueurs et des résultats entre le serveur et les clients.

La communication entre le client et le serveur est essentielle pour assurer le bon déroulement du jeu. Cette section analyse en détail les différentes fonctions responsables de la communication client-serveur, leur rôle et leur fonctionnement.

serverCommunication.c :

clientRegistration(gameData_t gameData) :

- Cette fonction gère l'enregistrement des clients en attendant leur connexion et en créant un nouveau thread pour chaque joueur.
- Elle initialise un thread d'écoute pour attendre les connexions entrantes des joueurs.
- Une fois qu'au moins un joueur est connecté, la fonction envoie le signal que le jeu peut commencer.
- Ensuite, pour chaque joueur connecté, un thread est créé pour gérer sa préparation et sa participation au jeu.

getPlayerChoice(gameData_t gameData, int playerIndex) :

- Cette fonction attend que le joueur spécifié envoie sa combinaison de couleurs au serveur.
- Une fois que le serveur reçoit la combinaison du joueur, elle est stockée dans les données du joueur pour être traitée ultérieurement.

sendResult(gameData_t gameData, int playerIndex) :
- Cette fonction envoie les résultats de la combinaison du joueur au client.
- Elle envoie également les résultats des autres joueurs au client afin que chaque joueur puisse voir les résultats des autres.

_listenningThreadHandler(void args) :
- Cette fonction gère le thread d'écoute du serveur qui attend les connexions entrantes des joueurs.
- Elle crée un nouveau thread pour chaque joueur connecté afin de gérer sa préparation et sa participation au jeu.

_clientReadyThreadHandler(void args) :
- Cette fonction gère le thread qui attend la confirmation de préparation d'un joueur.
- Elle attend que le joueur spécifié soit prêt à jouer, puis met à jour son statut de préparation dans les données du jeu.

Cette analyse fournit un aperçu détaillé du rôle de chaque fonction dans la communication client-serveur, facilitant ainsi la compréhension du fonctionnement global du jeu et de son architecture.

## 3.3 Structure applicative multi-thread

On utilise le diagramme suivant, l’objectif est d’avoir un thread par client afin de garder une disponibilité constante du serveur pour les clients :

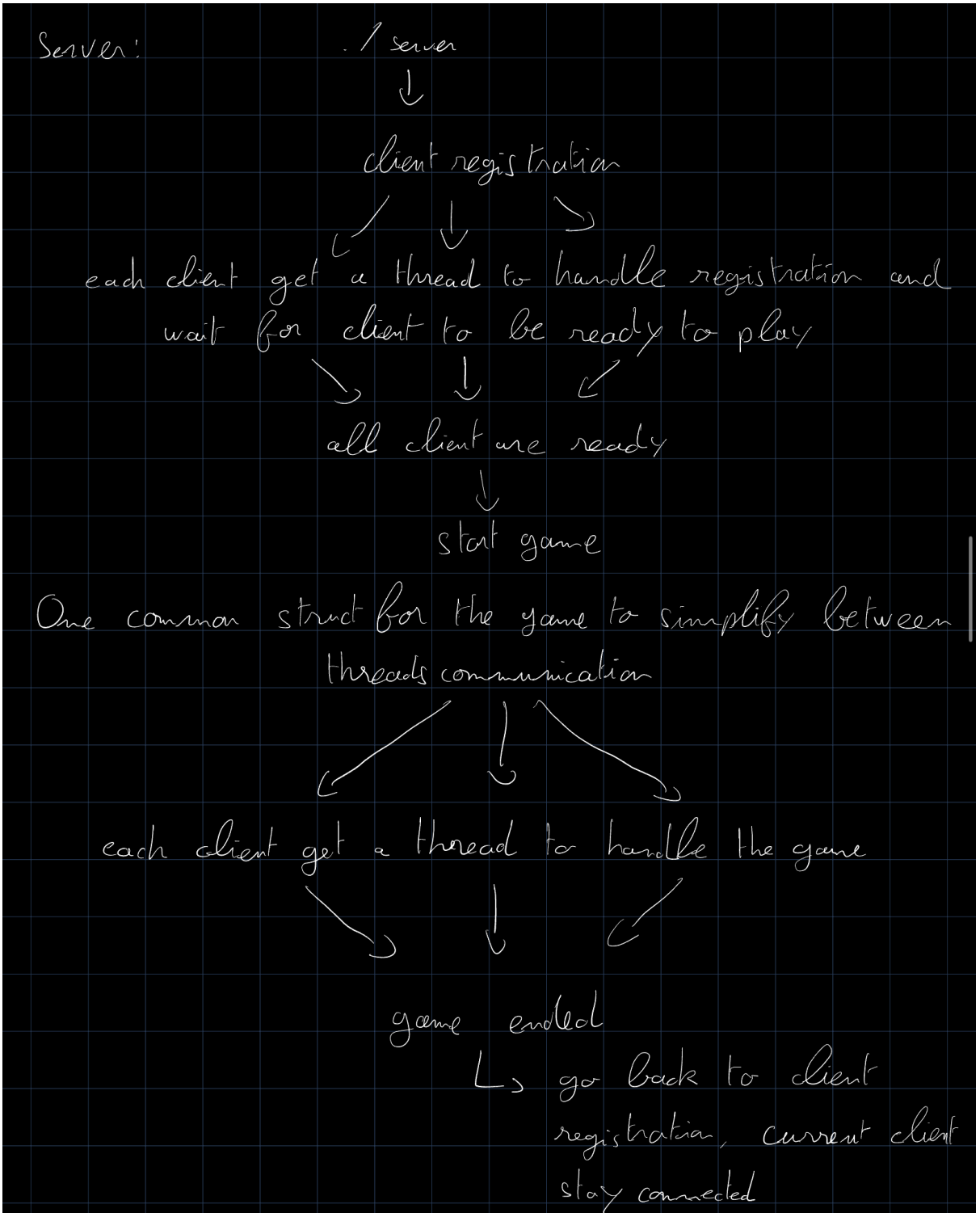


Figure 6 : Diagramme threads

## 3.4 Tests de validation

Voici les différents débuts de partie pour deux clients, ainsi que le lancement du serveur et les mises à jour après une combinaison :

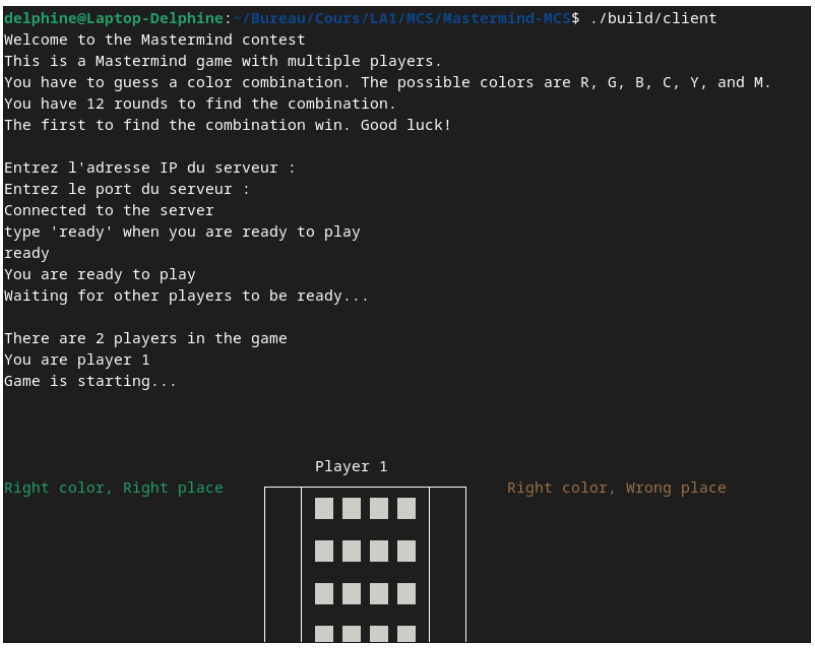


Figure 7 : Début de partie du client 1


Figure 8 : Début de partie du client 2

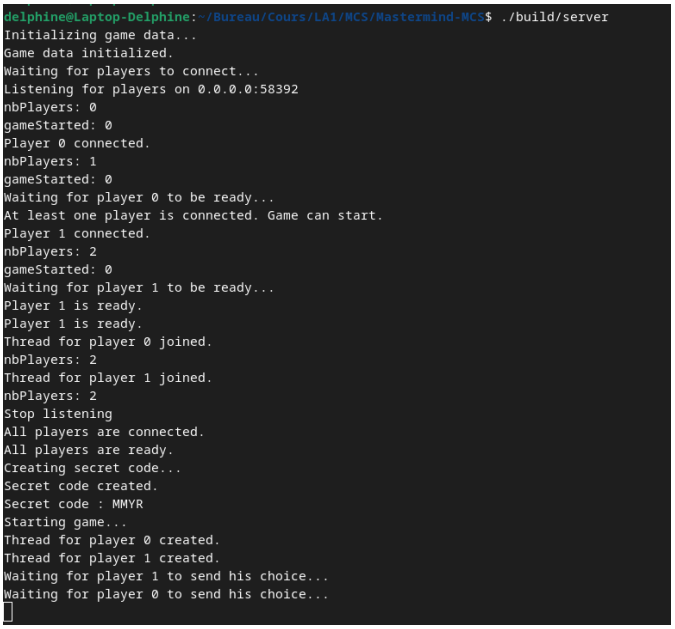


Figure 9 : Début de partie du serveur

On peut voir que pour le client 1 ou le client 2, ils arrivent à se connecter automatiquement au serveur, même lorsque l'adresse ip n'est pas rentrée pour le client 1, car l'adresse par défaut est 127.0.0.1 (adresse locale) et le port 58392.

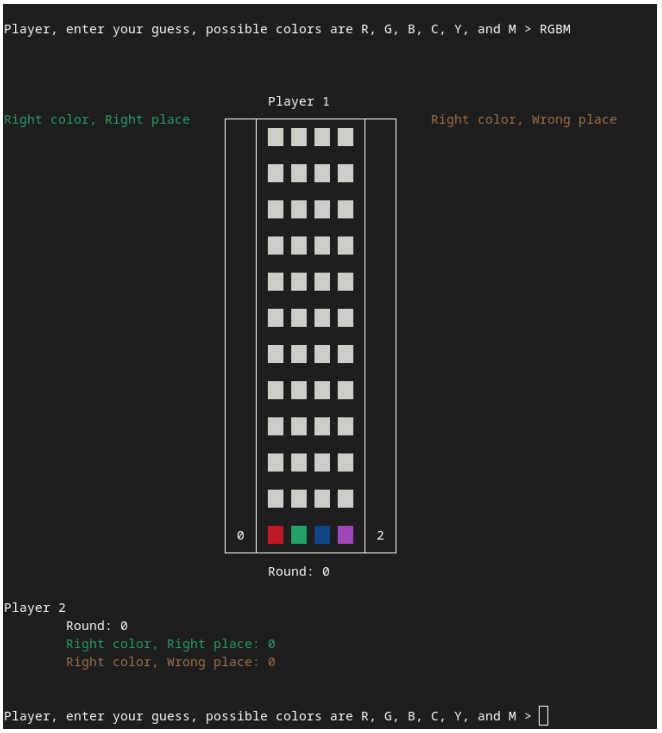


Figure 10 : Première combinaison du joueur 1

On voit ici que la combinaison tapée par le joueur 1 : RGBM est bien enregistrée et les couleurs apparaissent correctement et dans le bon ordre sur l'interface (représentée par les carrés).


Figure 11 : Affichage du serveur après le coup du joueur 1

On voit ici que le serveur a bien récupéré le coup du joueur 1 et a comparé la combinaison avec le code secret qu'il a généré : MMYR. Le nombre de bonne couleurs et de bonne place est également le même qui s'affiche sur l'IHM.

Le joueur ne peut qu'entrer les majuscules des lettres que nous lui avons mentionné en début de partie (voir photos début de partie). Ainsi, j'ai testé si la limitation du code fonctionnait correctement et si la combinaison était bien refusée lorsqu'une mauvaise lettre (donc couleur) était tapée ou lorsqu'une combinaison trop grande était proposée :

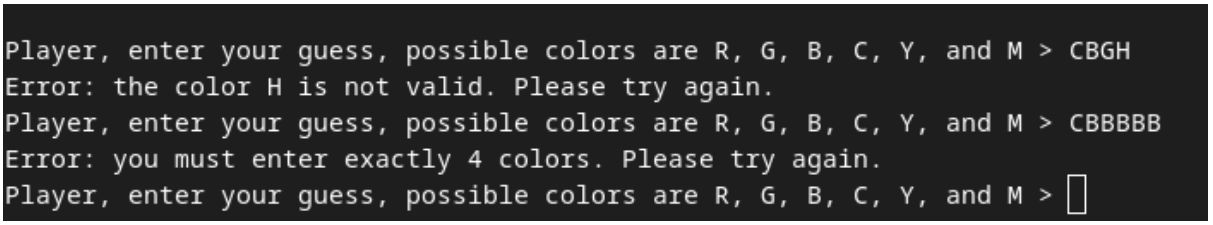


Figure 12 : Test de limitations des caractères entrées par le joueur

Nous voyons ici que le test est concluant.
Nous avons ensuite observé si le jeu du joueur 1 s'arrêtait au bout des 12 tentatives :

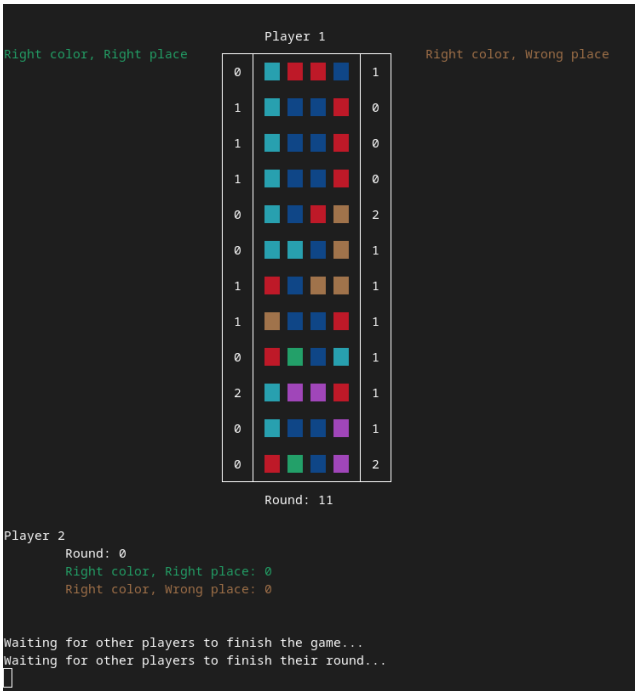


Figure 13 : fin du jeu au bout des 12 tentatives sans trouver le code

Nous avons ensuite testé avec le client 2 si lorsqu'il trouvait la combinaison secrète le jeu était terminée :


Figure 14 : Client 2 gagnant

Nous voyons ici que lorsque la combinaison secrète est trouvée, le jeu s'arrête. On voit également que les lignes d'information sur le client 1 (joueur 1, adversaire) sont bien mises à jour et donc que les clients communiquent correctement avec le serveur et vis versa.

Les tests que nous avons effectués sont donc concluants.

# PARTIE 4 : Bilan du projet

## 4.1 Conclusion sur le projet

Le projet MCS, axé sur la création d'un mastermind multijoueur, représente une réussite significative tant sur le plan conceptuel que technique. En se basant sur une spécification claire, le développement a été orienté vers la création d'une version interactive et collaborative du jeu de Mastermind, permettant à un groupe de joueurs de s'affronter pour trouver une combinaison secrète dans un laps de temps limité.

L'architecture du jeu repose sur un modèle client-serveur, où le serveur centralise la logique du jeu et coordonne les interactions entre les joueurs, tandis que les clients fournissent l'interface utilisateur. Cette structure permet une gestion efficace des parties, y compris la création, la gestion et la clôture des sessions de jeu.

La communication entre le serveur et les clients, basée sur des sockets Stream INET et le protocole TCP, garantit la fiabilité et l'ordonnancement des échanges de données, assurant ainsi une expérience de jeu fluide et synchronisée. De plus, l'utilisation de threads permet une gestion multi-utilisateur efficace, permettant à plusieurs joueurs de participer simultanément à une même partie sans compromettre les performances du serveur.

Sur le plan fonctionnel, le jeu offre une interface utilisateur conviviale et intuitive, permettant aux joueurs d'interagir facilement avec le jeu en proposant des combinaisons de couleurs et en recevant des informations sur leurs performances ainsi que sur celles des autres joueurs.

## 4.2 Difficultées rencontrées

Problème de synchronisation client-serveur :

Lors d’une première implémentation, nous n’avions pas vérifié à l’aide d’un acquittement que le client avait bien reçu les informations du serveur, il arrivait donc que le client ne lise que le dernier envoie du serveur, notamment sur la fin de la partie ou la synchronisation n’était pas assuré.

Multithreading et interruption de l’écoute :

pour la partie connexion du client et du serveur, nous avons eu quelque difficulté pour synchroniser les client et obliger les client déjà « ready » à attendre ce qui ne l’était pas. Une fois ce problème résolu, nous avons fait face à un autre problème. Lorsque tous les clients étaient prêts mais que le nombre de clients maximum n’était pas atteint, le thread dédié à l’écoute de connexion des nouveau client ne se fermait pas. Pour remédier à cela, nous avons dû simuler une connexion client depuis le serveur pour quitter l’appel de la

fonction listen des sockets. A l'aide d'un flag, on interrompt ensuite le thread automatiquement pour permettre le bon déroulement de la suite du programme.

On peut voir que la majorité de nos problèmes étaient liés au manque de maîtrise de certaines fonctions système comme les threads ou les sockets. Grâce à nos erreur sur ce projet, nous avons pu perfectionner nos connaissance et notre compréhension

## 4.3 Points d'améliorations possibles

Le projet MCS représente une réalisation notable dans le domaine des jeux multijoueurs, offrant une expérience interactive et stimulante aux joueurs. Toutefois, malgré le bon fonctionnement du jeu, plusieurs pistes d'amélioration peuvent être envisagées pour enrichir davantage l'expérience de jeu et renforcer l'attrait du projet pour les utilisateurs finaux.

Premièrement, la gestion des connexions et déconnexions des clients pourrait être optimisée pour garantir une expérience utilisateur plus fluide et transparente. La mise en place de mécanismes de reconnexion automatique en cas de défaillance de la connexion ou de déconnexion accidentelle des joueurs pourrait améliorer la robustesse et la convivialité du jeu.

Deuxièmement, l'introduction de fonctionnalités sociales telles que le choix des noms de joueur pourrait contribuer à créer un environnement de jeu plus engageant et personnalisé. Permettre aux joueurs de personnaliser leur identité virtuelle ajouterait une dimension sociale au jeu, favorisant ainsi l'interaction entre les participants et renforçant le sentiment d'appartenance à la communauté de joueurs.

En outre, l'ajout de fonctionnalités supplémentaires telles que des options de personnalisation de l'interface utilisateur, des modes de jeu alternatifs ou des défis spéciaux pourrait diversifier l'expérience de jeu et prolonger sa durée de vie. La création de nouveaux modes de jeu, tels que des tournois ou des épreuves chronométrées, pourrait également stimuler l'engagement des joueurs et encourager la compétition amicale entre les participants.

En conclusion, le projet MCS offre une base solide pour le développement continu d'un jeu multijoueur captivant et interactif. En explorant les possibilités d'amélioration mentionnées ci-dessus, nous pourrions consolider notre projet et créer une expérience de jeu encore plus enrichissante et divertissante pour les joueurs.

Git du projet : https://github.com/NexxFire/Mastermind-MCS