RAPPORT DE Réseau

OPTION **:**

Master en Systèmes Intelligents et Multimédia
Promotion : 26

Rédigé par :

- ADOSSEHOUN Kossi Josoué
- ALASSANI Saïb Gaston
- SANDANI Moyéme Jammel
- SAMO CHOUAKE Daniela

Encadrant:

- Nguyen Hong Quang

Année Académique : **2023-2024**

Introduction

Au sein de l'Institut Francophone International (IFI), au cours de l'année académique 2022-2023, le groupe 3 de la promotion SIM P26 a entrepris un projet ambitieux. Notre formation en informatique à l'IUT de Montpellier nous a conduit à plonger dans le monde complexe des applications client-serveur. Ces applications sont essentielles dans un éventail de secteurs, qu'il s'agisse de surveiller des machines à distance, d'effectuer des transactions financières via un téléphone, ou encore de permettre à des joueurs dispersés géographiquement de se connecter et de jouer en ligne.

Ce projet revêt un caractère exceptionnel, car il met en avant la création et l'implémentation d'un protocole. Notre objectif était de concevoir une application client-serveur, à partir de zéro, pour offrir une expérience de jeu en réseau unique. Plus précisément, notre défi consistait à développer une version en réseau du jeu classique du morpion (Tic-Tac-Toe), tout en intégrant la possibilité d'accueillir plusieurs spectateurs au cours des parties.

L'élément clé de notre projet est la mise en place d'un protocole dédié, qui orchestre la communication entre le client et le serveur, garantissant une expérience fluide et réactive pour les utilisateurs. Dans le cadre de cette présentation, nous décrirons en détail les besoins techniques et fonctionnels de notre projet à travers le cahier des charges. Ensuite, nous explorerons la logique générale de l'application, en mettant en avant la conception du protocole et les caractéristiques essentielles qui sous-tendent notre système.

La partie programmation de l'application fera l'objet d'une analyse approfondie, avec des extraits de code qui mettront en lumière le fonctionnement du protocole que nous avons mis en place. Enfin, nous dresserons un bilan des résultats obtenus, tout en adoptant un regard critique pour identifier les points forts et les opportunités d'amélioration qui pourraient enrichir davantage notre application.

Nous conclurons notre exposé en réfléchissant à l'organisation de notre projet, notamment en ce qui concerne la création du protocole et la collaboration au sein de notre groupe. En annexe, vous trouverez la totalité du code source de l'application, offrant une vue exhaustive de notre travail. Nous espérons que cette présentation vous plongera dans l'univers captivant des applications client-serveur, en mettant en avant la création et l'implémentation d'un protocole pour notre projet de morpion en réseau.

1. Cahier de charge

Le cahier des charges de notre projet se structure en trois sections distinctes. Tout d'abord, nous fournirons un aperçu du contexte du projet. Ensuite, nous détaillerons les diverses fonctionnalités destinées à l'utilisateur. Enfin, nous aborderons les contraintes techniques associées au projet.

1.1. Présentation du projet

Les applications client-serveur trouvent leur utilité dans de nombreux secteurs, qu'il s'agisse de contrôler des machines à distance dans l'industrie, d'effectuer des transactions financières directement depuis un téléphone dans le domaine financier, ou même de permettre à des joueurs de s'affronter simultanément, mais à distance, dans le domaine des jeux vidéo en ligne. Ce projet revêt une importance significative, car il nous offre l'occasion d'acquérir des compétences essentielles pour la création d'une application de base, compétences qui peuvent être potentiellement appliquées à d'autres projets.

Les éléments clés de ce projet résident dans la mise en place de la communication client-serveur, l'adaptation d'un jeu de morpion pour permettre une communication fluide, ainsi que la création d'une fonctionnalité de spectateur permettant de suivre en temps réel le déroulement de la partie.

1.2. Analyse des besoins fonctionnels

L'objectif principal de ce projet est de développer un jeu qui permettra à deux joueurs d'interagir entre eux en utilisant deux ordinateurs connectés via un réseau. Le jeu lui-même doit rester relativement simple, car l'accent principal réside dans la mise en place efficace de la communication réseau. Il est crucial de souligner que le morpion doit offrir la possibilité à deux joueurs de jouer à distance à travers deux ordinateurs distincts, tout en permettant également d'accueillir un public de spectateurs supplémentaires.

1.2.1. Règles du jeu implémenté

Le morpion est un jeu qui oppose deux joueurs sur une grille de 3 cases de largeur sur 3 cases de longueur. Le jeu se déroule de manière alternée, avec chaque joueur ayant son propre symbole, soit des ronds, soit des croix. Dans notre cas, nous avons choisi de laisser aux joueurs la possibilité de sélectionner leur symbole, en veillant à ce qu'aucun symbole ne soit répété. À chaque tour, un joueur remplit une case vide de la grille avec son symbole respectif. Le but du jeu est que le premier joueur parvienne à aligner trois de ses symboles, y compris en diagonale, pour remporter la partie. Si toutes les cases sont remplies sans qu'aucun joueur n'atteigne cet objectif, la partie se termine. Ce jeu est simple, mais il nécessite un échange significatif d'informations entre les joueurs, ce qui le rend adapté à notre projet. De nombreux jeux de plateau partagent un schéma similaire en termes d'échange de données, la complexité étant principalement liée aux règles du jeu en lui-même. Il est donc

envisageable que la mise en réseau de jeux tels que le Go ou les échecs suive un modèle similaire.

1.2.2. Diagramme d'activité

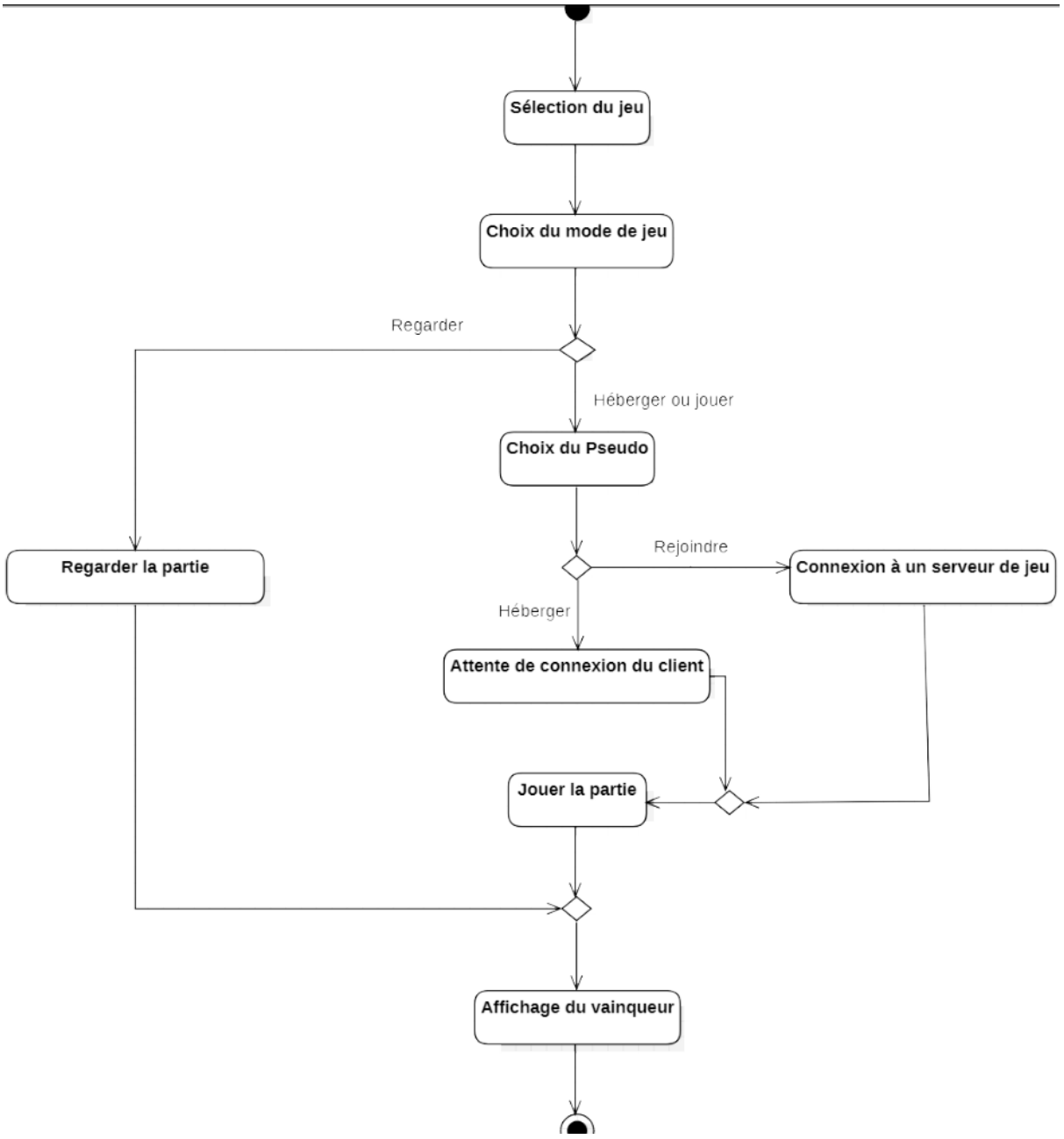

Figure 1.1 – Diagramme d'activité de l'application

1.3. Analyse des besoins non fonctionnels

Lorsque l'utilisateur lance l'application, il a le choix entre trois modes de jeu : le mode Spectateur, le mode Joueur Client (Rejoindre une partie), ou le mode Joueur Serveur (Héberger une partie). Dans tous les cas, les joueurs ont la

possibilité de personnaliser leur expérience en choisissant un pseudonyme et un symbole pour être identifiables par les autres participants.

Si un utilisateur opte pour le mode "Joueur Serveur", il est automatiquement inscrit dans la partie et attend qu'un autre joueur le rejoigne. En revanche, s'il sélectionne le mode "Joueur Client", il incarne le deuxième joueur et peut ainsi participer activement à la partie. Pour ceux qui préfèrent observer plutôt que jouer, le mode "Spectateur" permet de suivre le déroulement de la partie en cours, mais sans possibilité d'interagir avec elle. De plus, un spectateur peut rejoindre une partie en cours à tout moment.

Une fois que la partie est terminée, le vainqueur est clairement affiché pour que tous les participants puissent le voir.

2. Rapport technique

Dans ce rapport technique, nous entamerons notre exploration en plongeant dans la structure globale de notre application. Cette étape nous permettra de mettre en lumière la manière dont nous avons organisé le programme pour permettre à deux joueurs de s'affronter à distance, tout en offrant la possibilité à un public de spectateurs de suivre la partie.

Par la suite, nous détaillerons les différentes fonctions que nous avons conçues et intégrées, facilitant ainsi la compréhension du code et ouvrant la voie à une réutilisation éventuelle. Afin d'offrir une clarté supplémentaire, nous avons incorporé des diagrammes d'activité qui vous guideront étape par étape à travers le fonctionnement du programme.

Pour conclure, nous aborderons un diagramme des classes, à la fois pour la partie morpion et la partie réseau de notre application. Cela vous permettra de visualiser la structure de manière plus détaillée et d'appréhender les différentes composantes de notre projet.

2.1. Conception

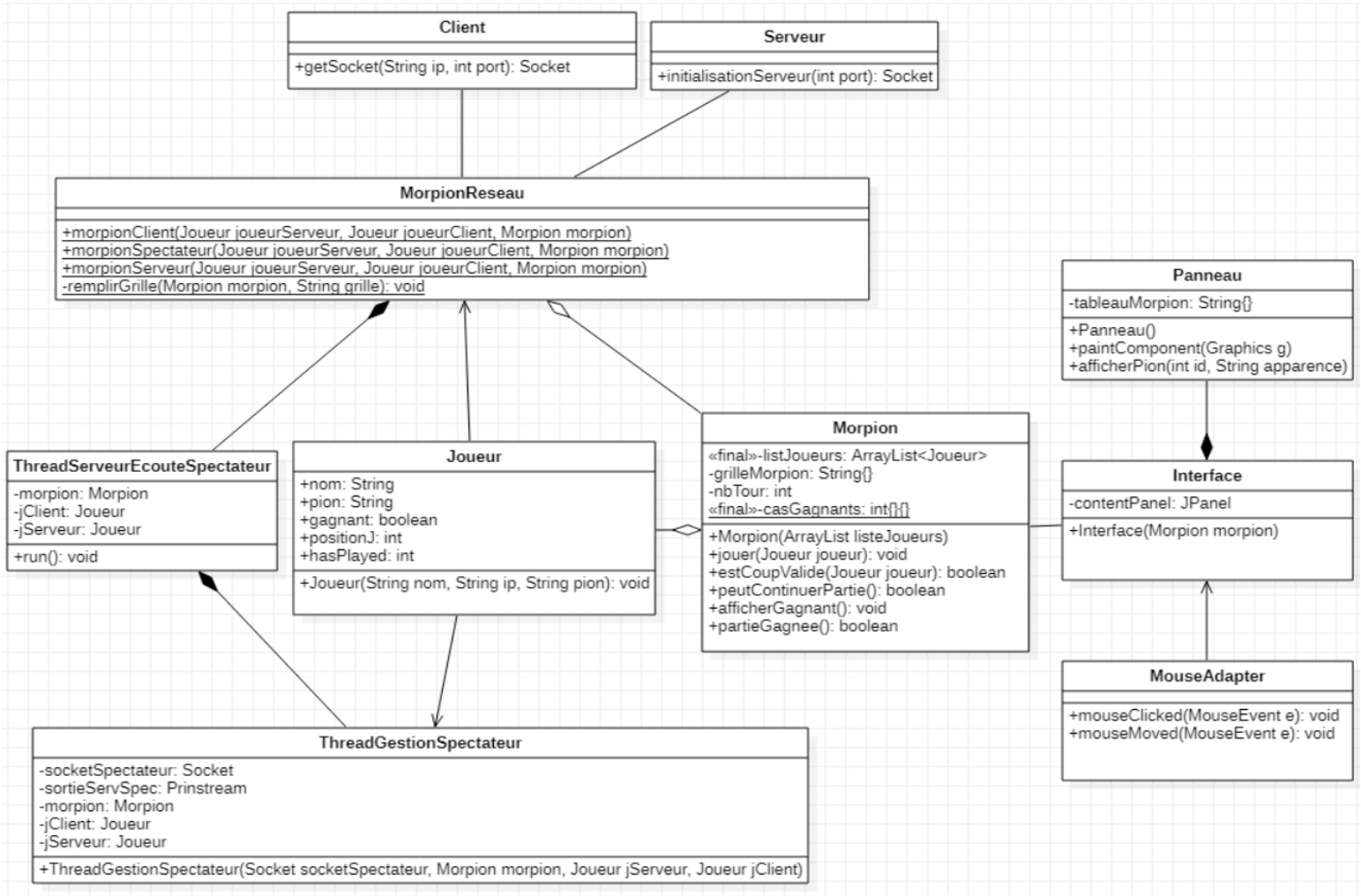

Figure 2.1 – Diagramme de classe

Dans la figure 2.1 ci-dessus, vous pouvez observer la structure de notre projet grâce à un diagramme de classe simplifié. Il est notable que la majorité des classes sont associées à MorpionReseau. En effet, MorpionReseau est doté d'un attribut de type Morpion, ce qui lui permet de superviser le fonctionnement du jeu Morpion pour l'utilisateur. De plus, pour gérer les spectateurs, MorpionReseau possède un attribut de type ThreadServeurEcouteSpectateur.

La classe Morpion, quant à elle, renferme une liste contenant les deux joueurs, ce qui lui permet de gérer les interactions entre le serveur et le client. En ce qui concerne la classe ThreadServeurEcouteSpectateur, elle crée des instances de ThreadGestionSpectateurs pour gérer de manière individuelle chaque spectateur.

2.1.1. Les Sockets

Dans le cadre de notre projet, nous avons opté pour l'utilisation de la technologie des sockets. Cette technologie permet d'établir des connexions via le protocole TCP (Transmission Control Protocol). Le protocole TCP est incontournable dans notre environnement numérique, ce qui justifie pleinement l'importance de son utilisation. Les sockets sont couramment employés chaque fois qu'il est nécessaire d'établir une communication entre plusieurs machines. La classe Socket, faisant partie de la bibliothèque Java avec l'interface "Closeable," s'est avérée particulièrement précieuse pour mettre en œuvre cette technologie qui joue un rôle essentiel dans notre projet. Notre intérêt principal porte sur la mise en place de la connexion entre deux machines, et c'est précisément là que cette technologie trouve son application.

2.1.2. Les Threads

Bien que les sockets soient indispensables pour établir une connexion entre deux machines, ils présentent des limitations lorsqu'il s'agit de gérer la connexion de multiples machines simultanément. Pour notre projet, nous devons autoriser plusieurs spectateurs à se connecter sur le même port d'un serveur unique. Malheureusement, pour manipuler efficacement et de manière autonome un grand nombre de connexions, il est essentiel de mettre en place des mécanismes de gestion indépendants. L'une des approches les plus élégantes et efficaces pour y parvenir consiste à utiliser des threads. En effet, les threads permettent de traiter de manière indépendante plusieurs fonctions, créant l'impression que les tâches s'exécutent en parallèle pour l'utilisateur. Vous pouvez observer le fonctionnement de plusieurs threads dans l'illustration fournie dans la figure 2.2 ci-dessous.

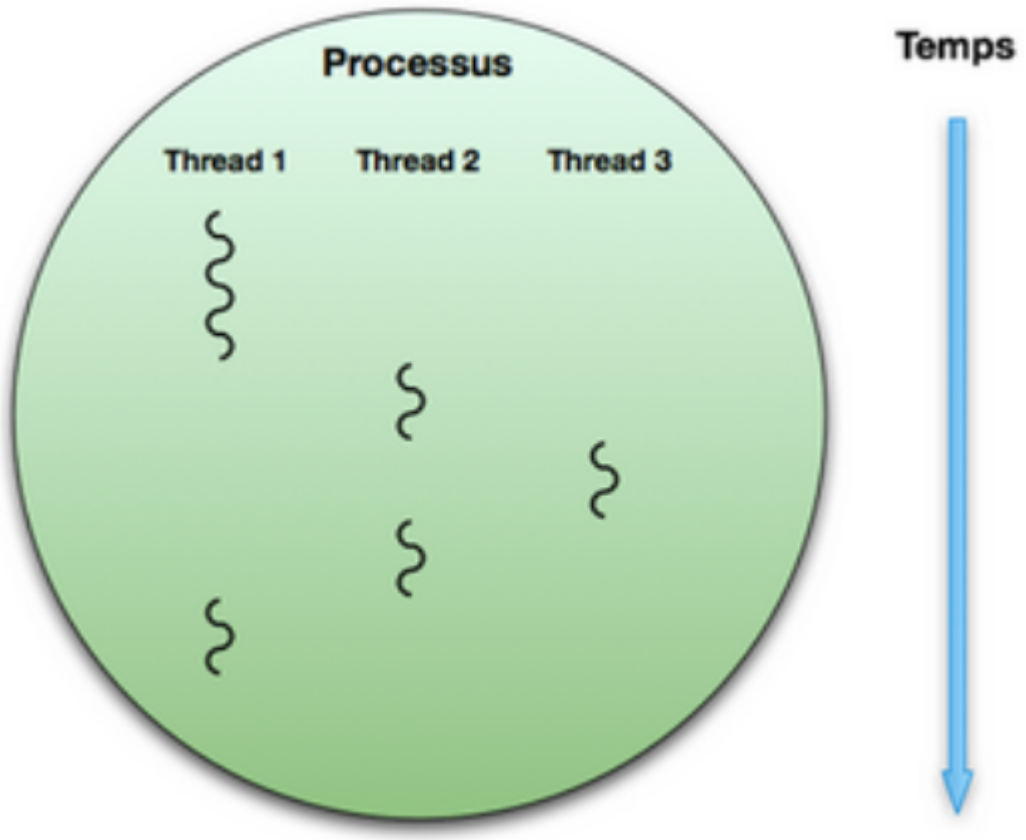

Figure 2.2 – Schéma explicatif thread

La puissance des threads nous permet de gérer théoriquement une infinité de spectateurs (dans la limite des capacités de la machine) se connectant au même serveur. La classe Thread de Java nous a donc été très utile dans l'implémentation de cette technologie.
Nous avons utilisé ces deux technologies au travers du langage Java. Nous avons choisi d'utiliser Java, car c'est le langage que nous avons le plus utilisé en cours, donc cela nous permet de plus facilement nous concentrer sur le projet, plutôt que l'apprentissage d'un langage que nous maîtrisons moins bien.

2.2. Réalisation

2.2.1. Le jeu

Comme mentionné précédemment, l'objectif principal de notre projet était de mettre en place une communication réseau entre plusieurs ordinateurs. Pour atteindre cet objectif, nous avons opté pour la création d'une version en réseau du jeu du morpion. Le choix d'un jeu simple s'est avéré judicieux, car il nous a permis de nous concentrer plus efficacement sur la mise en œuvre de la communication réseau, tout en offrant une application concrète et ludique de cette technologie. Ci-dessous, vous trouverez la présentation de la classe Morpion.

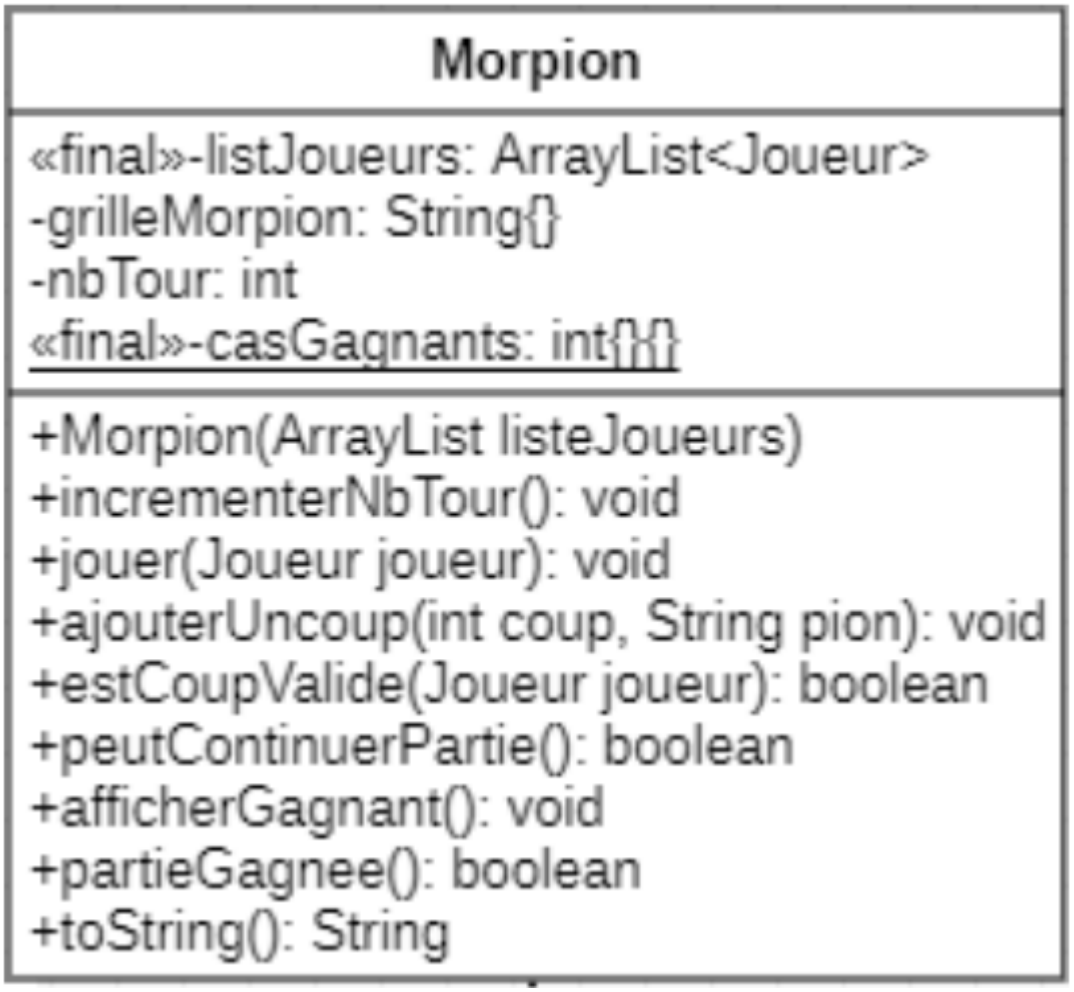

Figure 2.3 – Classe Morpion

Nous allons débuter en présentant les attributs de la classe. Nous disposons d'une liste de joueurs destinée à contenir les deux joueurs impliqués dans le jeu, ainsi qu'un tableau de chaînes de caractères nommé "grilleMorpion," qui sert à enregistrer les emplacements des pions des joueurs. Il y a également un entier "nbTour," explicitement désigné pour suivre le nombre de tours joués. Enfin, pour évaluer les configurations gagnantes, nous avons un tableau d'entiers nommé "casGagnants," qui regroupe toutes les configurations victorieuses. Maintenant que nous avons couvert les attributs, nous allons explorer les différentes méthodes.

La méthode "jouer" prend un joueur comme argument et attend la position du pion qu'il a choisi. Ensuite, avec la fonction "estCoupValide," nous vérifions que le coup du joueur respecte les règles du jeu, puis nous l'inscrivons dans la grille. La fonction "estCoupValide" s'assure que le coup est réalisable, en vérifiant qu'il se situe dans la grille et que l'emplacement choisi n'est pas déjà occupé. La méthode "peutContinuerPartie" renvoie un booléen indiquant si la partie est terminée. Par exemple, la partie se termine si le nombre de tours atteint 9 ou si la fonction "partieGagnee" renvoie "true."

La fonction "partieGagnee" est responsable de la vérification d'un gagnant. Si le nombre de tours est inférieur à trois, il ne peut y avoir de gagnant, et elle renverra "false." Dans le cas contraire, la fonction parcourt la grille "grilleMorpion" et le tableau "casGagnants" pour vérifier si un des joueurs est dans une configuration de victoire, c'est-à-dire si 3 de ses pions sont alignés. Si tel est le cas, elle renverra "true."

En ce qui concerne la méthode "toString," elle suit le format suivant :

Figure 2.4 – Résultat du toString

2.2.2. Le réseau

Dans la composante réseau de notre projet, nous avons créé deux classes distinctes : une classe "Client" et une classe "Serveur."

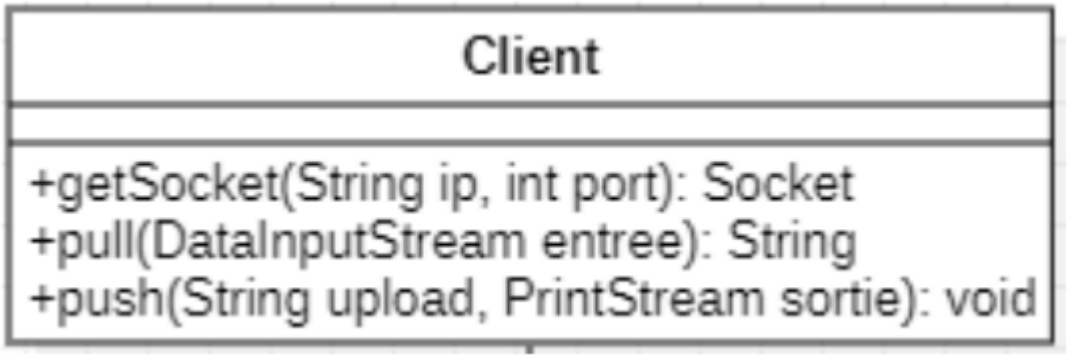

Figure 2.5 – Classe client

La classe "Client" se compose exclusivement de méthodes statiques. Elle a pour rôle de créer un socket avec une adresse spécifiée, et elle offre également la possibilité d'envoyer et de recevoir des données grâce aux fonctions "push" et "pull."

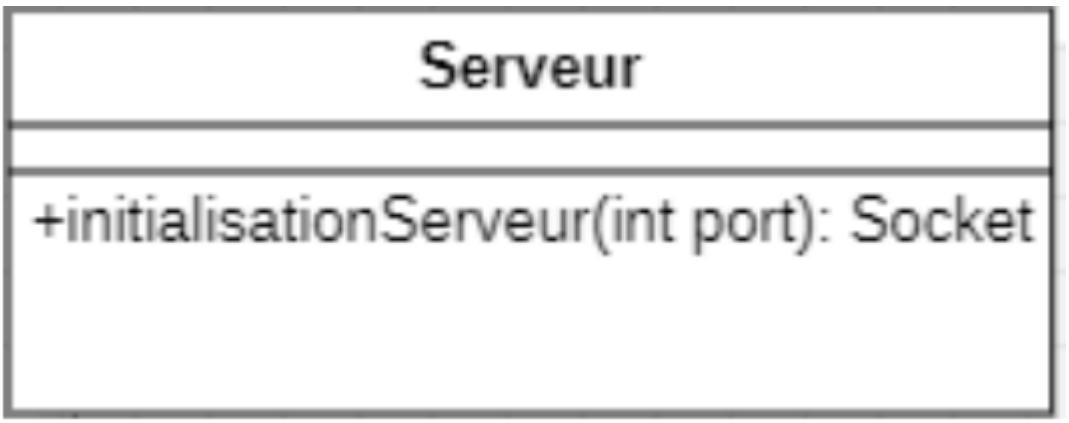

Figure 2.6 – Classe serveur

La classe "Serveur" est responsable de la création d'un socket et de l'écoute sur ce dernier. Pour ce faire, elle tire parti des méthodes de la classe "Client."

2.2.3. La gestion du jeux en réseau

2.2.3.1. Le client

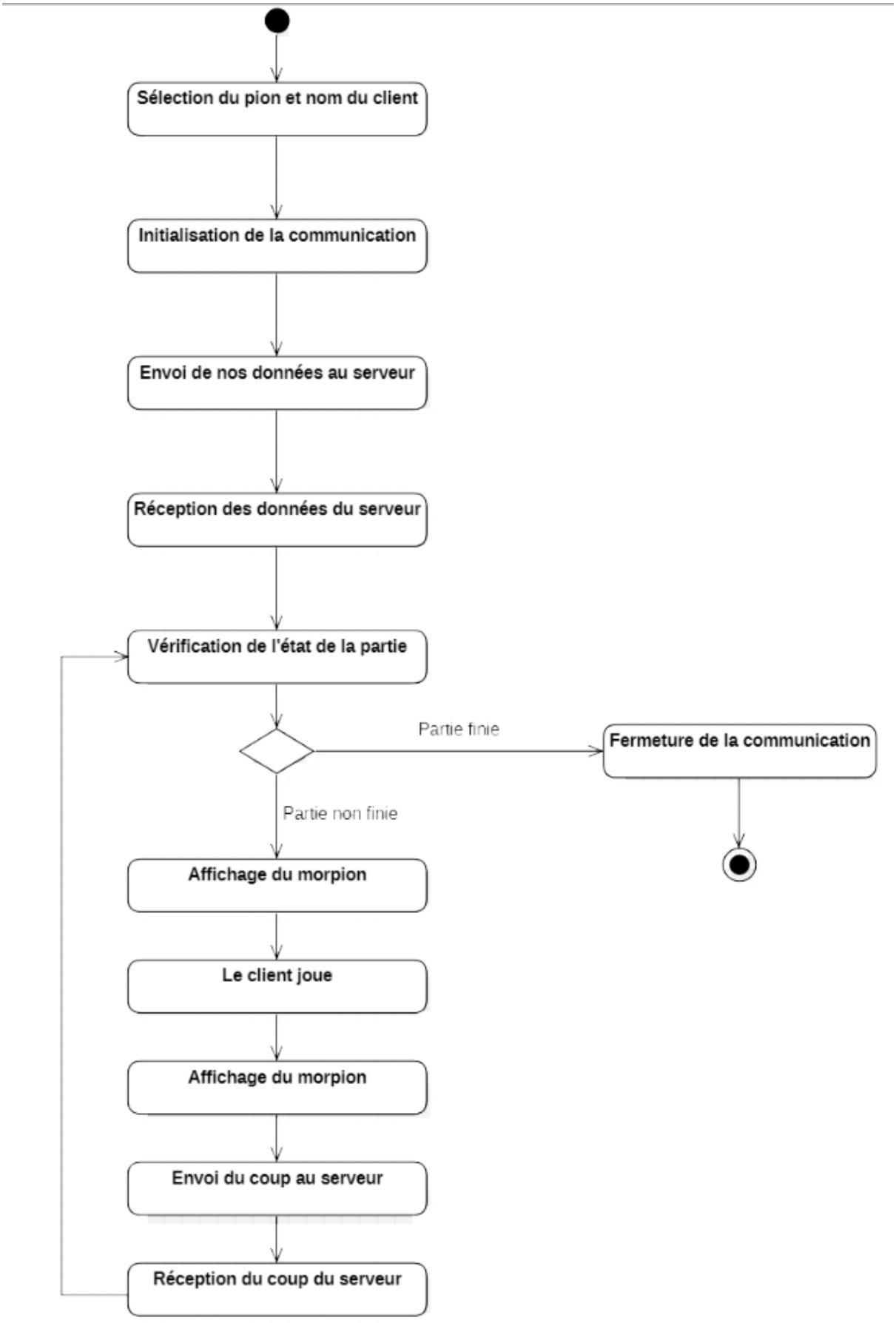
Sélection du pion et nom du client
Initialisation de la communication
Envoi de nos données au serveur
Réception des données du serveur
Vérification de l'état de la partie
Partie finie
Fermeture de la communication
Partie non finie
Affichage du morpion
Le client joue
Affichage du morpion
Envoi du coup au serveur
Réception du coup du serveur

Figure 2.7 – Diagramme d’activité du client

Dans la figure 2.7 ci-dessus, nous avons représenté le fonctionnement du client. La fonction "morpionClient" est une méthode de la classe "MorpionRéseau." Le client débute en fournissant ses informations, à savoir son pseudonyme et son choix de symbole. Ensuite, il s'établit une connexion avec le serveur et transmet ces mêmes informations au serveur. L'envoi de ces données permet au serveur de reconnaître le client pour le reste de la partie.

Une fois ces informations échangées, l'interface du morpion s'affiche, et le client entame son tour de jeu. Après chaque coup joué, l'interface du morpion est réactualisée, et le client envoie le coup qu'il vient de jouer au serveur. Cette alternance d'affichage est cruciale pour maintenir une interface à jour à chaque coup, qu'il provienne du client ou qu'il soit reçu du serveur.

Ensuite, le client attend de recevoir le coup du serveur, et ce processus se répète jusqu'à la fin de la partie. La fonction dédiée à déterminer si la partie est achevée permet de différencier une partie nulle d'une victoire d'un des joueurs. Elle offre une méthode fiable pour décider si le jeu peut se poursuivre ou non, évitant ainsi d'attendre inutilement un coup du serveur si ce dernier s'est déconnecté et a terminé sa partie.

2.2.3.2. Le spectateur

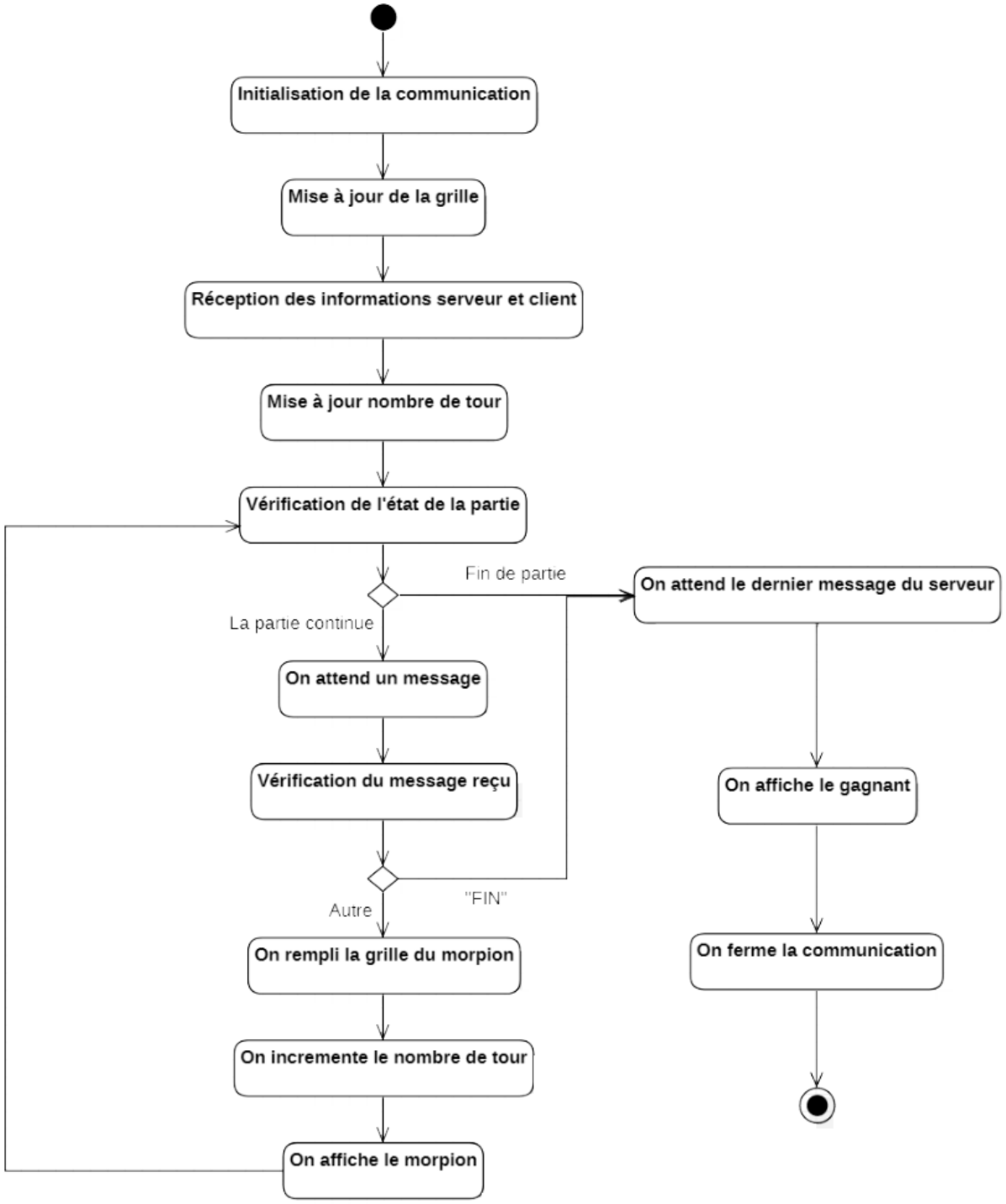

Figure 2.8 – Diagramme d'activité du spectateur

Dans notre programme, les spectateurs ne sont pas tenus de fournir un pseudonyme, car nous avons délibérément choisi de les rendre invisibles pour les joueurs actifs. Au lieu de cela, lorsqu'un spectateur rejoint une partie, nous recevons de l'information telle que la grille, les données des deux joueurs et le nombre de tours déjà effectués.

À partir de ce point, nous entrons dans une boucle qui ressemble à celle du joueur-client. Cela signifie que la mise à jour du morpion a lieu à chaque coup. Cependant, au lieu d'enregistrer uniquement le coup joué, nous mettons à jour la grille complète par souci de simplicité. Avant de mettre à jour la grille, nous vérifions toujours si le message reçu du serveur est "FIN." Ce message indique que le serveur a mis fin à la partie, ce qui nous permet d'éviter les erreurs liées au fait que le serveur n'est plus disponible, alors que le spectateur attend la suite de la partie. Si la partie est terminée, nous attendons un dernier message du serveur, affichons le nom du gagnant, puis clôturons la communication. L'ensemble de ces messages reçus du serveur permet de maintenir une synchronisation avec lui, évitant ainsi des erreurs dues à une déconnexion soudaine.

2.2.3.3. Le serveur

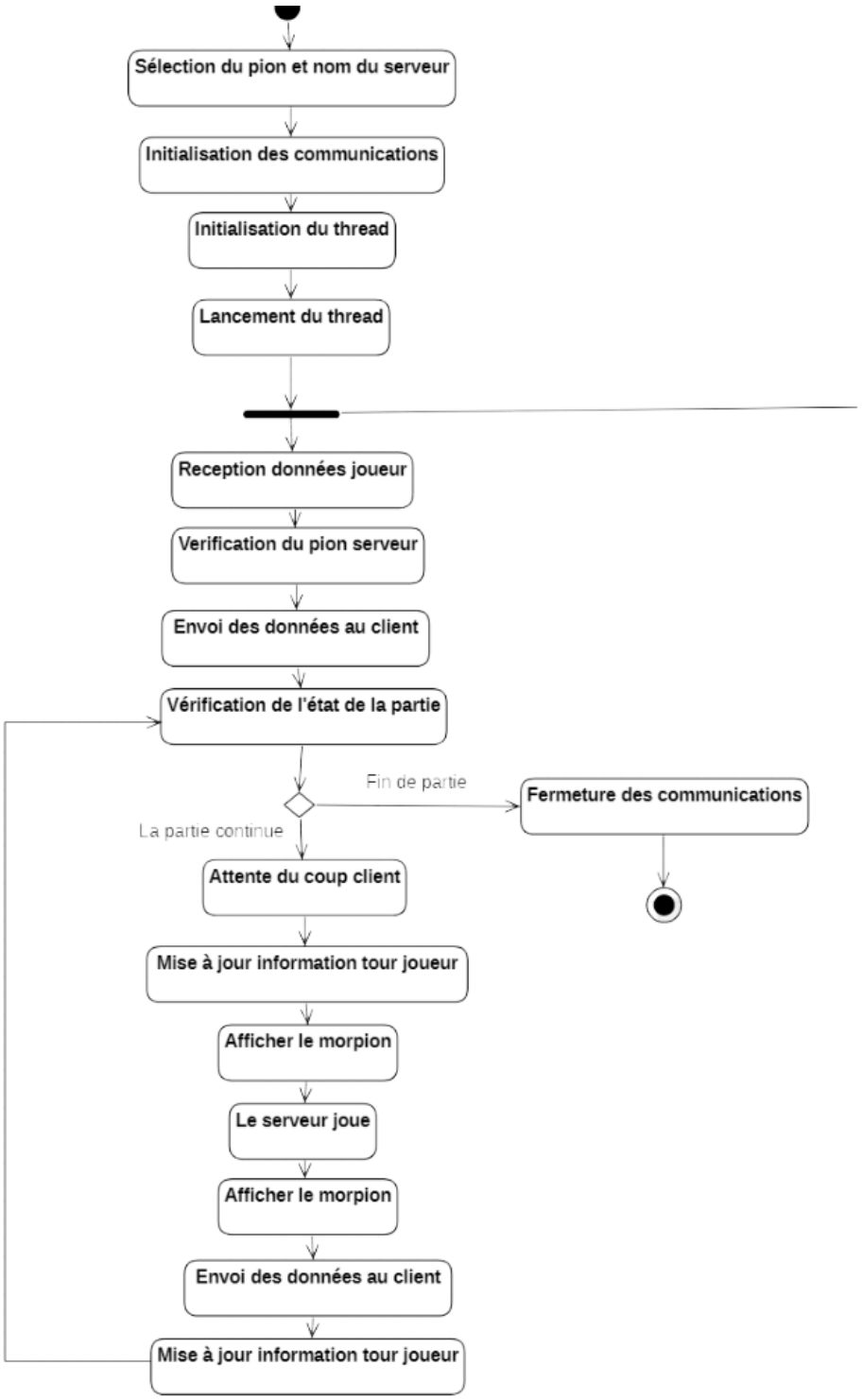

Figure 2.9 – Diagramme d'activité du serveur principal

Le début de la méthode du serveur présente des similitudes marquées avec celle du client : l'utilisateur saisit ses informations, puis nous initialisons la communication avec le client et le spectateur. Cependant, le spectateur nécessite le démarrage d'un thread, que nous détaillerons ultérieurement. La séquence qui suit est pratiquement en miroir de la partie client. En effet, elle commence par attendre le coup du client, et à ce moment, elle met à jour le fait

que le client a joué. Cette mise à jour s'avère utile pour le thread du spectateur. Par la suite, la méthode alterne entre l'envoi du coup, l'affichage du morpion et la mise à jour du tour des joueurs, répétant ces étapes jusqu'à ce que la partie soit terminée.

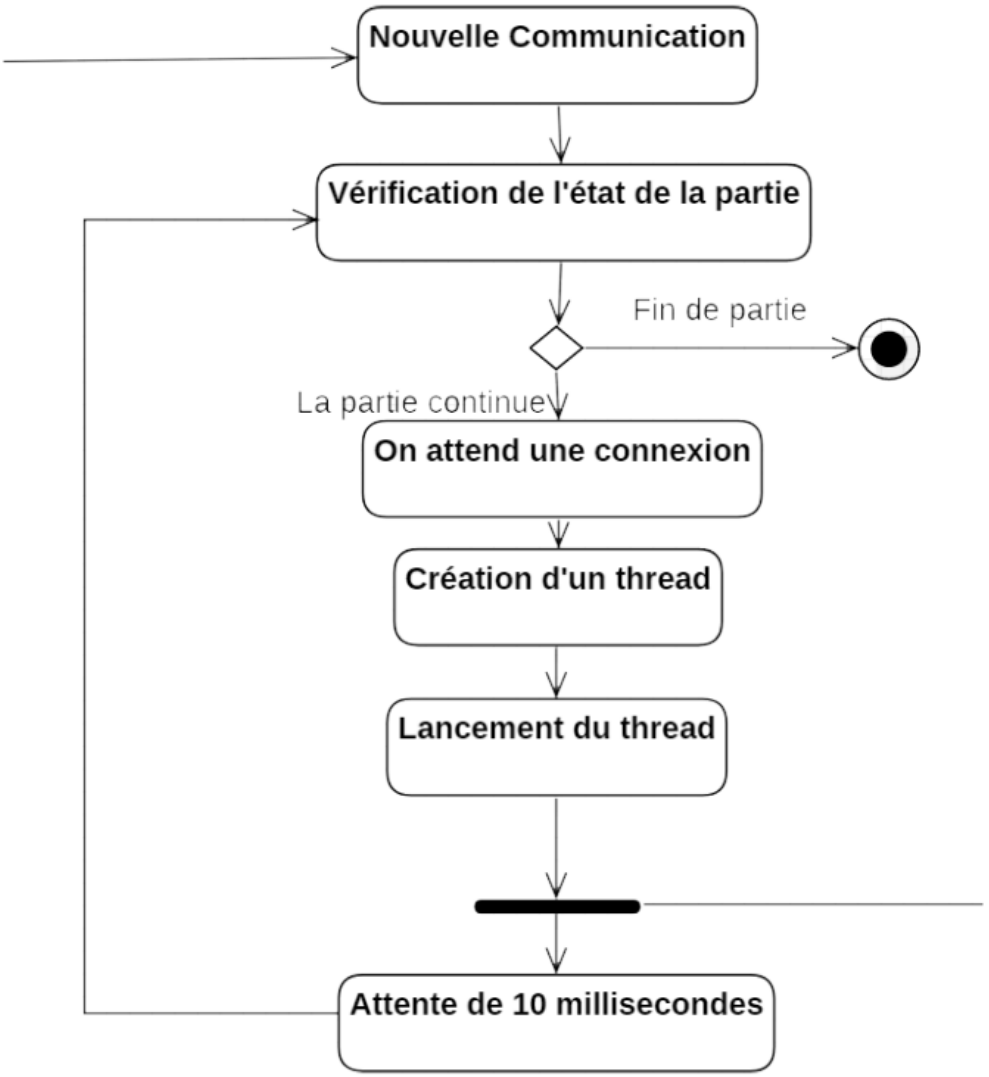

Figure 2.10 – Diagramme d’activité du thread d’écoute

Passons maintenant à l'analyse du diagramme d'activité du thread d'écoute du spectateur. Dans ce schéma, le serveur d'écoute est en attente d'un nouveau spectateur. Lorsqu'un spectateur se connecte, le programme initie un nouveau thread de gestion du spectateur, et cela se répète tant que la partie n'est pas achevée.

```java
@Override
public void run() {

        try {
            ServerSocket s_ecoute = new ServerSocket( port: 2001);
            while (morpion.peutContinuerPartie()) {
                Socket spectateur = s_ecoute.accept();
                System.out.println("Connexion d'un nouveau spectateur...");
                Thread t = new Thread(new ThreadGestionSpectateur(spectateur, morpion, jServeur, jClient));
                t.start();
                Thread.sleep( l: 10);
            }

        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
}
```

Figure 2.11 – Extrait de code du thread d'écoute

Au tout début de ce thread, un socket est initialisé au port 2001, spécialement réservé aux spectateurs. Il est à noter que nous demeurons dans la boucle "While" tant que le morpion n'est pas terminé, c'est-à-dire que la fonction "morpion.peutContinuerPartie()" renvoie "true." Par la suite, nous accédons à la fonction bloquante "accept()," qui est chargée de créer un socket pour les spectateurs dès qu'un utilisateur se connecte à ce port spécifique.

Après la création de ce socket spectateur, nous instaurons immédiatement un nouveau thread qui englobe ce socket, le morpion et les joueurs. Ce thread est ensuite lancé sans délai. Sa principale responsabilité est de gérer le spectateur, et nous détaillerons davantage cette gestion dans le schéma suivant. Ce code met en exergue la puissance des threads, qui, en théorie et dans la limite des capacités de la machine, permettent de connecter un nombre potentiellement infini de spectateurs.

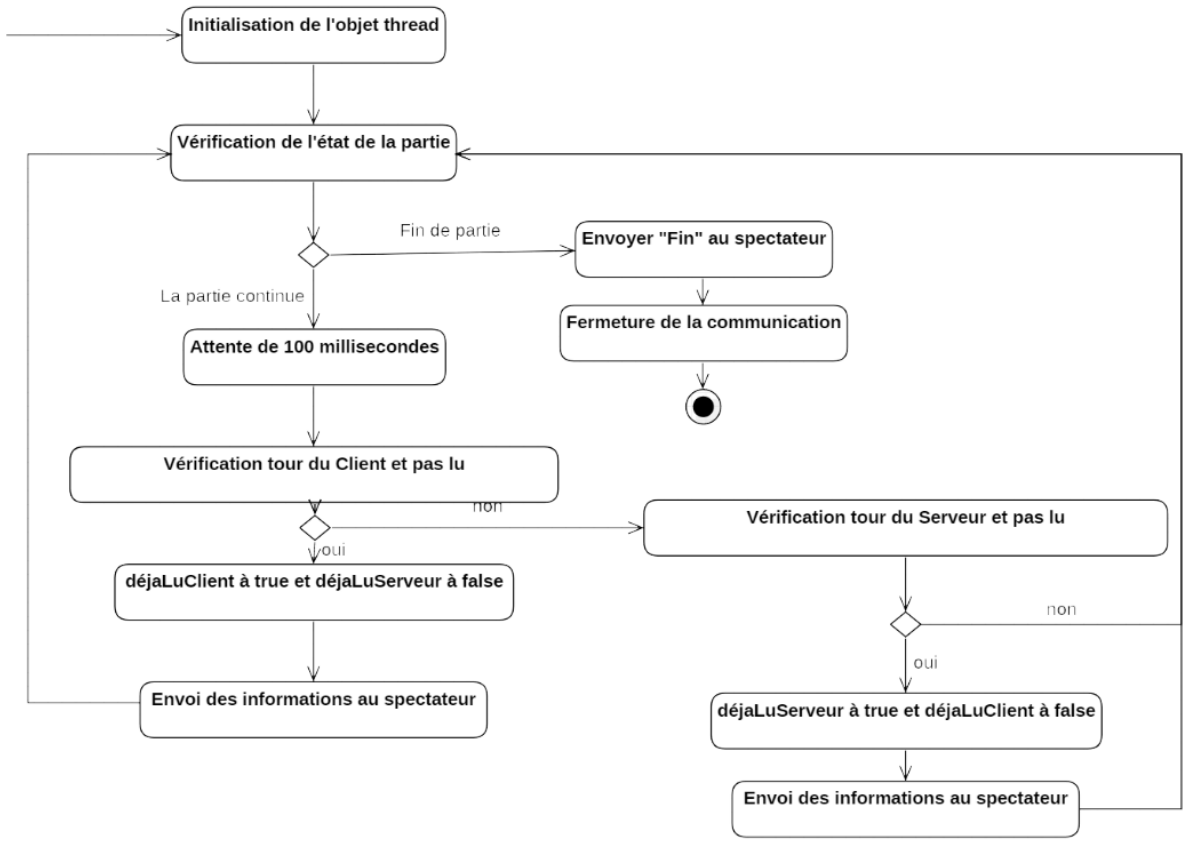

Figure 2.12 – Diagramme d'activité du thread gestion spectateur

2.2.3.4. L'interface graphique

Pour améliorer l'expérience des utilisateurs, nous avons opté pour la création d'une interface graphique permettant de jouer au morpion. Cette interface devient accessible une fois que la connexion entre les deux joueurs a été établie. Elle se présente sous la forme d'une fenêtre affichant une grille numérotée, où chaque case correspond à une case du morpion, prête à recevoir des coups de jeu.

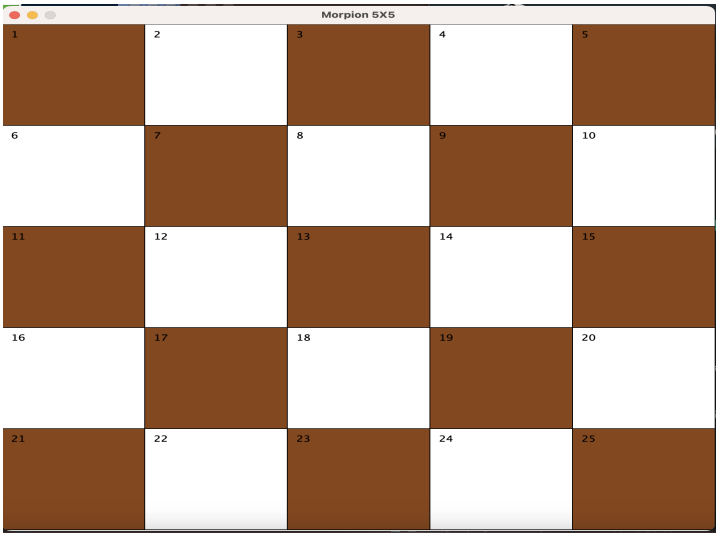

Figure 2.13 – Affichage du morpion dans l’interface graphique

2.2.3.5. Validation, résultats et perspectives

Pour tester notre programme, nous avons utilisé l'outil de débogage intégré à IntelliJ IDEA, qui nous permet de parcourir le code pas à pas. Pour l'évaluation de la partie réseau, nous avons fait appel à des affichages dans le terminal, ce qui nous a permis de surveiller l'état des variables en circulation sur le réseau et de vérifier la bonne exécution des échanges. Cette approche s'est révélée plus pratique, notamment en présence de threads susceptibles de modifier les variables de manière parfois difficilement prévisible. De plus, le débogueur n'était pas idéal pour vérifier la connexion réseau.

En ce qui concerne les résultats et les perspectives, nous avons réussi à implémenter la majorité des fonctionnalités spécifiées dans le cahier des charges. Voici un aperçu des fonctionnalités que nous avons réussi à réaliser :

- Nous avons mis en place un jeu de morpion fonctionnel.
- Ce jeu permet aux utilisateurs de jouer en ligne les uns contre les autres.
- Il est possible d'avoir au moins un spectateur qui peut observer la partie, et notre application peut gérer plusieurs spectateurs, bien que nous n'ayons pas testé le nombre maximal.

Comme pour tout projet, il reste des possibilités d'amélioration pour enrichir notre application au-delà des exigences de base. Certaines de ces améliorations sont de nature mineure, tandis que d'autres pourraient nécessiter une refonte plus significative du projet. Parmi les améliorations possibles, nous pourrions envisager les points suivants :

- Améliorer l'affichage de l'interface graphique en intégrant davantage d'éléments tels que le nombre de tours et les demandes de saisie (Le lancement du serveur, le choix des joueurs, des spectateurs, la saisie du nom du joueur en interface et le choix du pion).
- Mettre à jour la grille au bon moment, plutôt que lorsque la souris est déplacée.
- Refactoriser le projet pour le rendre adaptable à d'autres jeux, en créant une interface de jeu générique. Cette approche rendrait le projet plus évolutif en facilitant l'ajout de nouveaux jeux.
- Permettre de rejouer une nouvelle partie chaque fois qu’une partie se termine.
- Permettre la création de plusieurs salles de jeux pour permettre à plusieurs joueurs de jouer
- Afficher les scores des joueurs.

3. Les guides

3.1. Guide d’installation

Pour récupérer le projet, vous devrez accéder à GoogleClassRoom et télécharger le fichier zippé puis ouvrez dans IntelliJ après l’avoir dézippé.

Pour utiliser le projet, il est nécessaire d'installer IntelliJ IDEA et d'exécuter la classe "Main" Si vous souhaitez jouer en mode multijoueur avec des personnes locales, vous devrez saisir l'adresse IP de la personne qui sera l'hôte du jeu.

Pour jouer en mode multijoueur en ligne, une personne doit faire office d'hôte et doit configurer son routeur pour ouvrir les ports nécessaires. Cela permettra aux clients disposant de la bonne adresse IP et du bon port de se connecter à distance.

3.2. Guide d’utilisation

3.2.1. Démarrer le programme

Si vous n’avez pas le fichier exécutable, voici la démarche à suivre.

Figure 3.1 – Lancer l’application

Après avoir ouvert le projet dans IntelliJ IDEA, rendez-vous dans la classe "Main" que vous pouvez apercevoir ci-dessus.

Pour lancer le programme, il vous suffira de cliquer sur le bouton en forme de triangle vert. Cette action déclenchera la compilation du programme et son exécution.

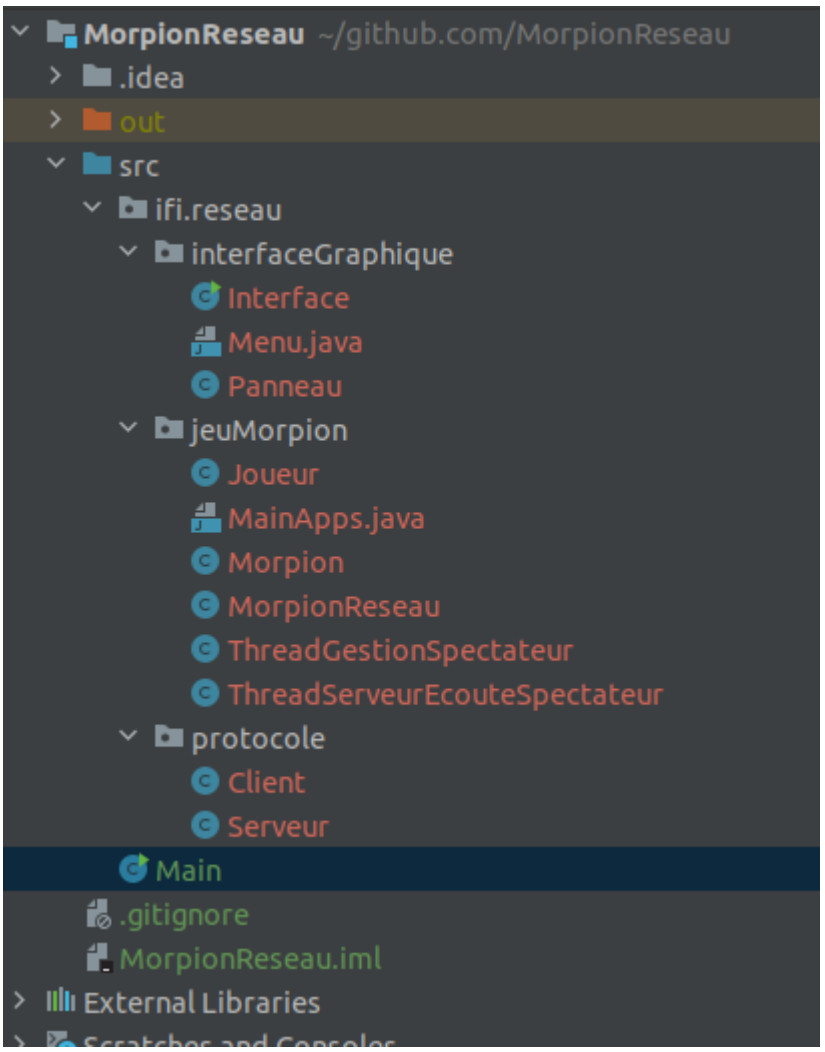

Figure 3.2 – Tutoriel compilation

3.2.2. Choix du mode de jeu

Lorsque vous avez démarré le programme, vous trouverez l'écran ci-dessous :

```
1=Jouer
2=Regarder
3=Héberger
4=Quitter
```

Figure 3.3 – Choix du mode de jeu

- Si vous souhaitez être l'hôte d'une partie pour jouer avec un autre joueur, saisissez "3" dans le terminal, puis appuyez sur "Entrée".
- Si vous désirez rejoindre une partie déjà hébergée par quelqu'un, entrez "1".
- Si votre intention est de regarder une partie en cours entre deux joueurs, saisissez "2".
- Pour quitter la partie et arrêter le programme, entrez "4".

3.2.3. Mode Joueur

3.2.3.1. Saisie des informations

Si vous optez pour le jeu ou l'hébergement d'une partie, l'expérience sera très similaire. Le programme vous invitera à saisir votre nom et votre choix de symbole :

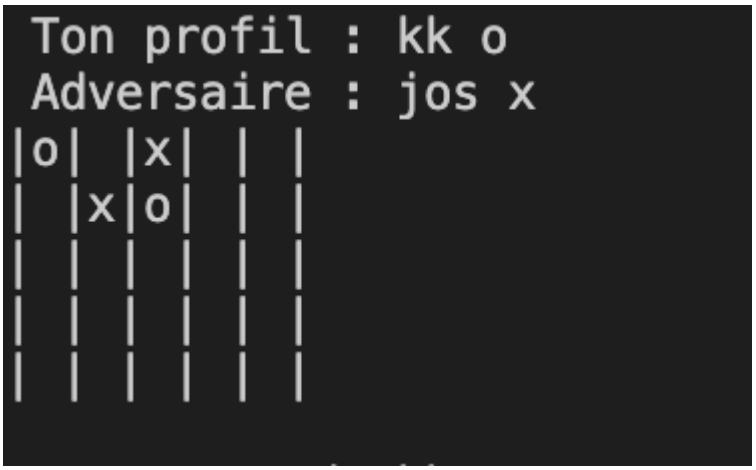

Figure 3.4 – Choix du nom et pion

Par défaut, l'hôte est désigné par le nom "Serveur" et le symbole "O," tandis que le client est identifié par le nom "Client" et le symbole "X." Il est important de noter qu'il ne faut pas inclure d'espaces ni de caractères multiples lors de la saisie du symbole, car le programme n'acceptera pas de telles données et vous demandera de les saisir à nouveau.

3.2.3.2. Jouer une partie

Si la fenêtre du morpion que vous voyez ci-dessous n'apparaît pas en premier plan, assurez-vous de la mettre en premier plan, car c'est la fenêtre dans laquelle vous allez jouer.

Pour jouer, vous devez patienter jusqu'à ce que le client effectue son coup si vous êtes l'hôte. Si vous êtes le client, vous pouvez cliquer sur la case de votre choix pour y placer votre pion.

En cas de problème avec l'affichage des pions, placez votre souris sur l'interface

3.2.4. Mode Spectateur

Si vous avez opté pour le mode spectateur, vous pouvez observer la partie directement dans le terminal ou via l'interface graphique. Si vous rencontrez des problèmes d'affichage, essayez de placer la souris sur la fenêtre pour résoudre ces problèmes.

3.2.5. Le match nul

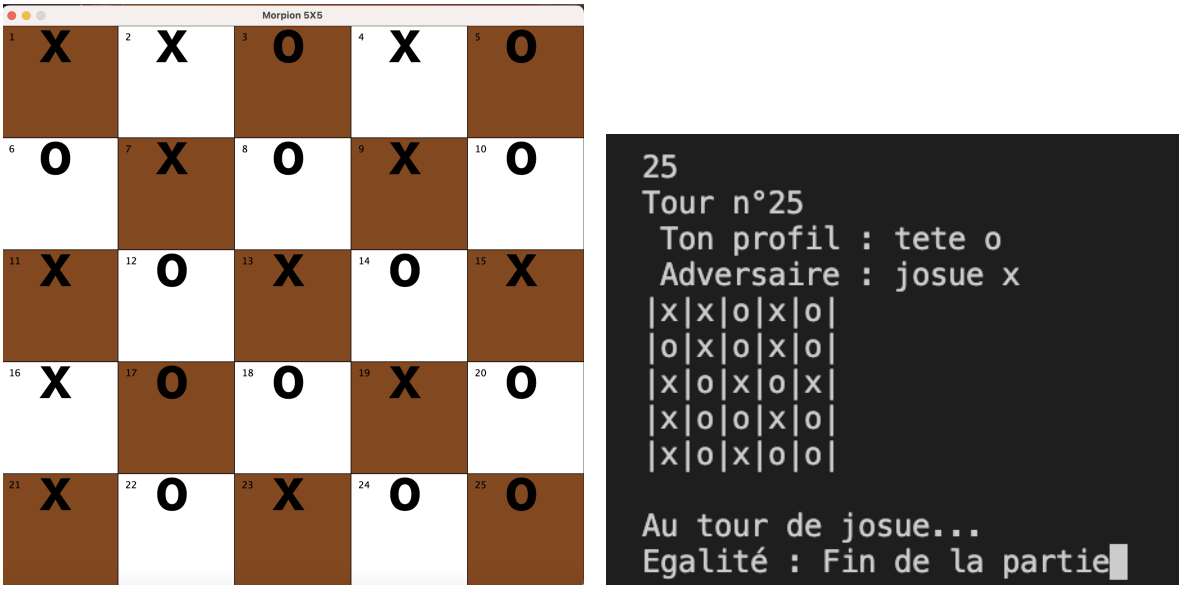

Figure 3.6 – Match nul

3.2.6. Fin de la partie

Lorsque la partie se termine, cela signifie qu'un joueur a aligné trois pions ou qu'il ne reste plus de cases vides. À ce moment, la partie se clôture, et vous pouvez examiner son état dans le terminal, comme illustré ci-dessous.

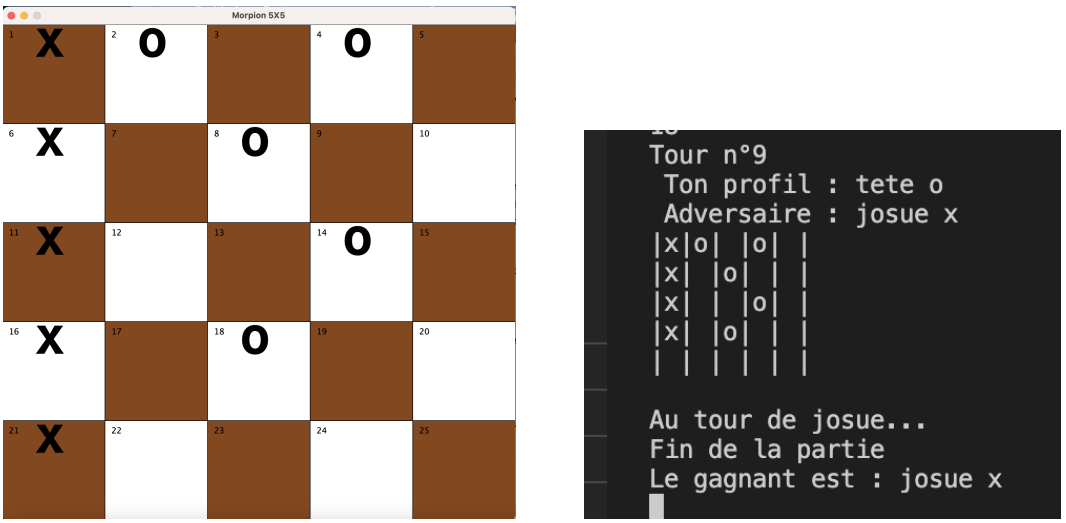

Figure 3.6 – Fin de la partie

Conclusion

En conclusion, ce projet réalisé dans le cadre de notre formation en informatique à l'Institut Francophone International a permis la création d'une application de jeu de morpion avec des fonctionnalités de jeu en réseau et de spectateur. L'objectif principal était de mettre en place une communication client-serveur robuste et de permettre à deux joueurs distants d'interagir tout en offrant la possibilité à plusieurs spectateurs de suivre la partie.

L'application propose une interface graphique conviviale, offrant une expérience utilisateur agréable. Elle permet aux utilisateurs de jouer en tant qu'hôte ou client, de visionner des parties en cours et de se connecter à distance pour jouer en multijoueur.

Le projet a été testé avec succès, bien que certaines fonctionnalités supplémentaires n'aient pas pu être mises en œuvre, telles que la sélection de jeux disponibles et de parties en cours. Des améliorations potentielles incluent l'intégration de plus d'informations dans l'interface graphique, des mises à jour en temps réel de la grille de jeu, et la possibilité d'adapter le projet à d'autres jeux grâce à une structure plus générique.

Dans l'ensemble, ce projet a permis d'acquérir des compétences essentielles en matière de programmation réseau, de gestion de threads, et de développement d'interfaces graphiques. Il sert de base solide pour d'autres projets de jeux multijoueurs et constitue une étape importante dans notre formation en informatique.