Jeu en réseau
Rapport de projet tuteuré

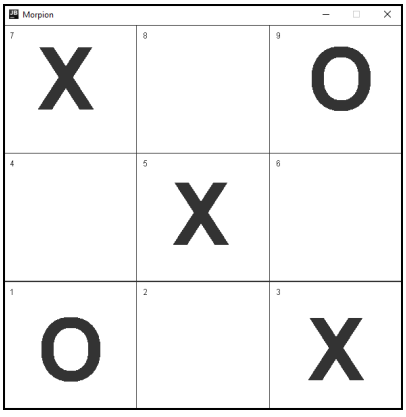
Tuteur de projet : Francis Garcia

Erwan LE GOFF, Clément GUITTENY, Adrien CORN, Florent LE MAIRE

Université de Montpellier

Diplôme Universitaire Technologique d'Informatique

Année Spéciale



2019-2020

Remerciements

Nous tenons à remercier notre tuteur M. Francis Garcia, qui nous a permis de réaliser ce projet et de nous orienter tout au long de sa conception.

Nous remercions également les membres du département Informatique de Montpellier qui nous ont permis d'avoir accès à des logiciels tels que l'IDE IntelliJ ainsi que la plateforme GitHub grâce auxquels nous avons pu travailler en équipe de manière efficace et organisée.

Sommaire

Table des figures

Glossaire

Socket : D'après Wikipédia, "un socket est un élément logiciel qui est aujourd'hui répandu dans la plupart des systèmes d'exploitation. Il s'agit d'une interface logicielle[...], grâce à laquelle un développeur exploitera facilement et de manière uniforme les services d'un protocole réseau." Le socket permet donc de facilement gérer la connexion entre plusieurs processus, entre deux machines distantes, ou sur la même machine.

Thread : Programme se déroulant en parallèle et assimilable à un fil d'exécution. Les threads permettent à l'application de gérer plusieurs utilisateurs en simultané. Lorsque plusieurs utilisateurs exécutent le programme ou lui envoient des demandes, un fil d'exécution est créé et géré pour chacun.

Introduction

Ce projet est réalisé dans le cadre de notre formation en année spéciale d'informatique à l'IUT de Montpellier. Les applications client-serveur sont utilisées dans de nombreux secteurs, que ce soit dans l'industrie pour contrôler des machines à distance, dans les finances pour réaliser des transactions directement à partir du téléphone, mais aussi dans les jeux vidéo en ligne, pour que plusieurs joueurs se situant à distance puissent jouer au même jeu en simultané.

Afin de comprendre au mieux la logique derrière le fonctionnement de ce type d'applications, nous avons décidé de réaliser une application client-serveur, en partant de zéro.

Le but du projet est donc de réaliser un morpion, auquel il est possible de jouer en réseau, en incluant la possibilité d'accueillir plusieurs spectateurs au cours de la partie.

Après avoir présenté les besoins techniques et fonctionnels au travers du cahier des charges, nous présenterons la logique générale de l'application, avant de nous intéresser à son application plus concrète en termes de programmation. Nous parcourrons différents extraits du code afin de voir l'essentiel du fonctionnement de la partie réseau de notre application. Nous exposerons ensuite les résultats avec un regard critique, de manière à faire ressortir les points positifs, mais aussi les améliorations possibles qui pourraient enrichir notre application. Nous ferons par la suite une rétrospection sur l'organisation de notre projet. Enfin, vous trouverez en annexe, parmi d'autres documents, la totalité du code de l'application afin d'en avoir une vue exhaustive.

1. Cahier des charges

Le cahier des charges est scindé en trois parties, en premier, nous présenterons le contexte du projet, ensuite nous expliquerons les différentes fonctionnalités à direction de l'utilisateur, pour enfin, exposer les contraintes techniques associées au projet.

1.1 Présentation du projet et contexte

Les applications client-serveur sont utilisées dans de nombreux secteurs, que ce soit dans l'industrie pour contrôler des machines à distance, dans les finances pour réaliser des transactions directement à partir du téléphone, mais aussi dans les jeux vidéo en ligne, pour pouvoir jouer au même jeu, en simultané, mais à distance. Ce projet est donc important, car il nous permet d'apprendre à réaliser une application de base, potentiellement applicable à d'autres projets.

Les éléments clés sont la communication client-serveur, l'adaptation d'une application Morpion pour qu'elle communique, mais aussi la mise en place d'un spectateur qui peut suivre la partie en temps réel.

1.2 Analyse des besoins fonctionnels

L'objectif de ce projet est donc de coder un jeu grâce auquel deux joueurs pourront interagir à travers deux ordinateurs connectés en réseau. Le jeu doit donc dans un premier temps être simple, l'aspect réseau étant la partie la plus importante du projet. Il est important d'insister sur le fait que le morpion doit permettre à deux joueurs d'interagir à travers deux ordinateurs distants, avec la possibilité d'accueillir plusieurs spectateurs.

1.2.1 Rappel des règles du morpion

Le morpion est un jeu opposant deux joueurs sur une grille de 3 cases de largeur et 3 cases de longueur. Le jeu se déroule au tour par tour et chaque joueur possède respectivement des ronds ou des croix. Dans notre cas, nous avons choisi de laisser la possibilité de laisser au joueur de choisir son pion, sans doublon possible. À chaque tour, le joueur remplit une case vide par le symbole lui correspondant. Le premier joueur à aligner trois de ses symboles (diagonales comprises) gagne la partie. S'il ne reste

plus de case, la partie est finie. C'est donc un jeu simple, mais échangeant suffisamment d'informations entre les joueurs pour correspondre aux attentes de notre projet. Beaucoup de jeux de plateau échangent des données similaires, la complexité se faisant principalement sur le jeu en lui-même, nous pouvons donc facilement en conclure que la partie en réseau sera similaire pour des jeux comme Go ou bien les échecs.

1.2.2 Diagramme d'activité

Vous pouvez vous référer à la figure 1.1 pour la représentation graphique de notre analyse fonctionnelle. Nous allons présenter et expliquer nos choix concernant ce qu'il est possible de réaliser par l'utilisateur. Nous décrirons étape par étape son fonctionnement et nos choix.

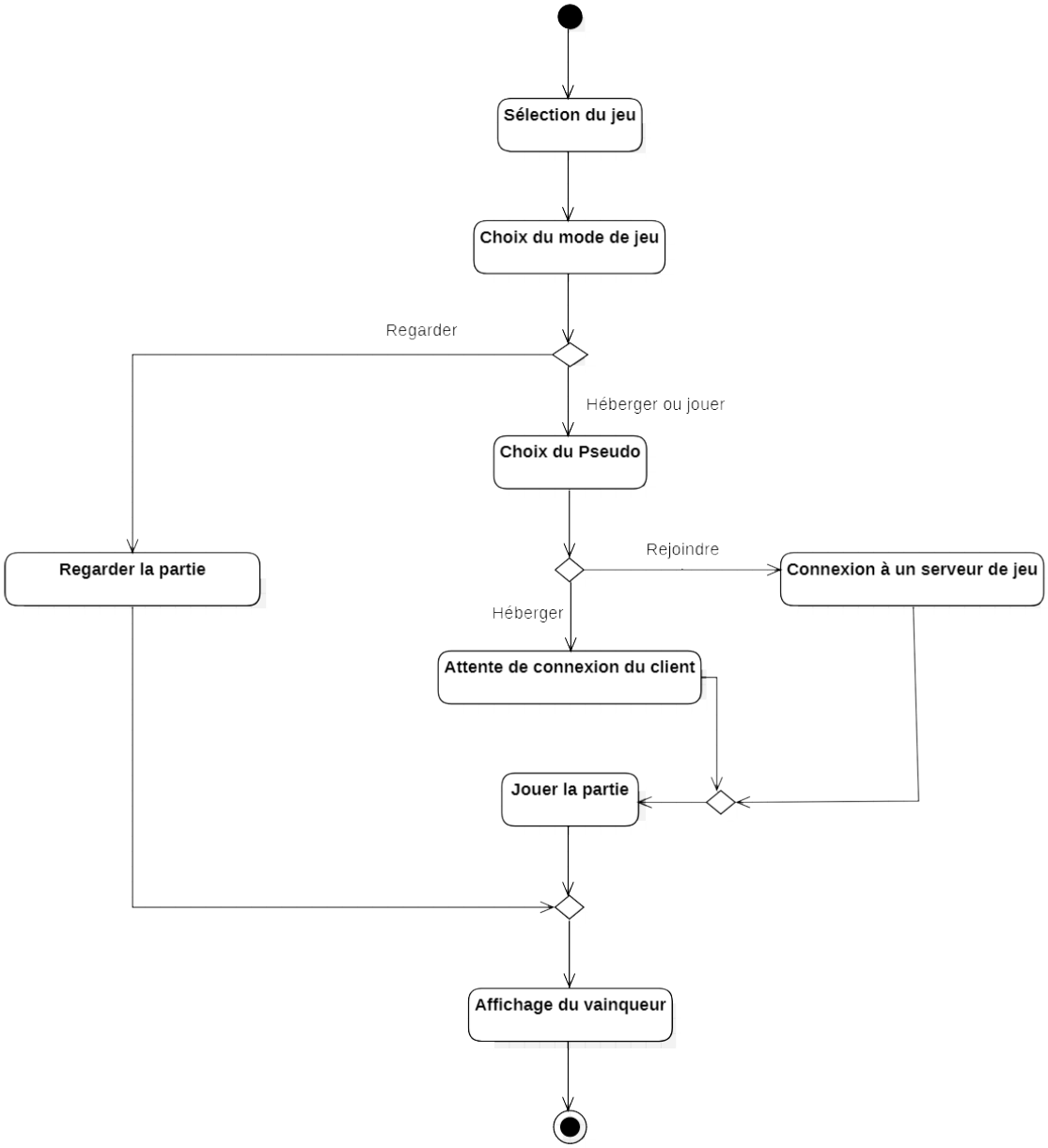


FIGURE 1.1 – Diagramme d'activité de l'application

L'utilisateur commence par sélectionner le mode de jeu : Regarder (Spectateur), rejoindre une partie (Joueur Client) ou héberger une partie (Joueur Serveur). Les joueurs peuvent choisir un pseudo et un pion afin d'être identifiables par les autres utilisateurs. L'hébergeur est obligé de participer à la partie, et il attend un autre joueur. S'il choisit « Joueur », il peut alors incarner le second joueur et participer à la partie. S'il choisit « Spectateur », la partie lui est alors retransmise, mais il ne peut pas interagir. Un spectateur doit pouvoir rejoindre une partie en cours à tout moment. Une fois la partie finie, le vainqueur est affiché.

1.3 Analyse des besoins non fonctionnels

Le langage de programmation utilisé sera le Java. Nous avons fait ce choix, car ce langage est celui avec lequel notre équipe est le plus familiarisée et qu'il nous donne accès à tous les éléments nécessaires à la réalisation de notre projet.

Notre application doit fonctionner sur PC, à la demande de notre tuteur, mais aussi, car nous sommes plus à l'aise avec cette plateforme. Nous avons choisi de la programmer en Java, car nous avons plus d'expérience avec ce langage, cela permet donc de mieux se concentrer sur les problématiques liées au réseau.

Notre environnement de travail repose sur l'IDE IntelliJ et le site d'hébergement GitHub. Nous avons choisi IntelliJ IDEA car il est très performant pour la programmation Java, de plus, c'est celui que nous avons le plus étudié en POO, enfin il est disponible à l'IUT. Nous hébergeons notre projet sur GitHub, car c'est un outil qui a déjà fait ses preuves, et dont nous possédions tous déjà des identifiants.

Pour la partie réseau, nous devons utiliser les sockets, car c'est l'une des technologies les plus utilisées dans les applications client-serveur.

Les threads seront nécessaires pour pouvoir gérer plusieurs spectateurs simultanément.

2. Rapport technique

Dans ce rapport technique, nous commencerons par explorer la logique globale de notre application, c'est-à-dire la façon dont nous avons organisé le programme afin de faire jouer deux joueurs en réseau, avec la possibilité d'accueillir de multiples spectateurs. Ensuite nous expliciterons les différentes fonctions créées et implémentées afin de simplifier la compréhension du code et sa possible réutilisation. Pour aider à la compréhension, nous avons également ajouté des diagrammes d'activité afin de suivre étape par étape le déroulement du programme. Pour finir, nous aborderons un diagramme des classes de la partie morpion ainsi que la partie réseau.

2.1 Conception

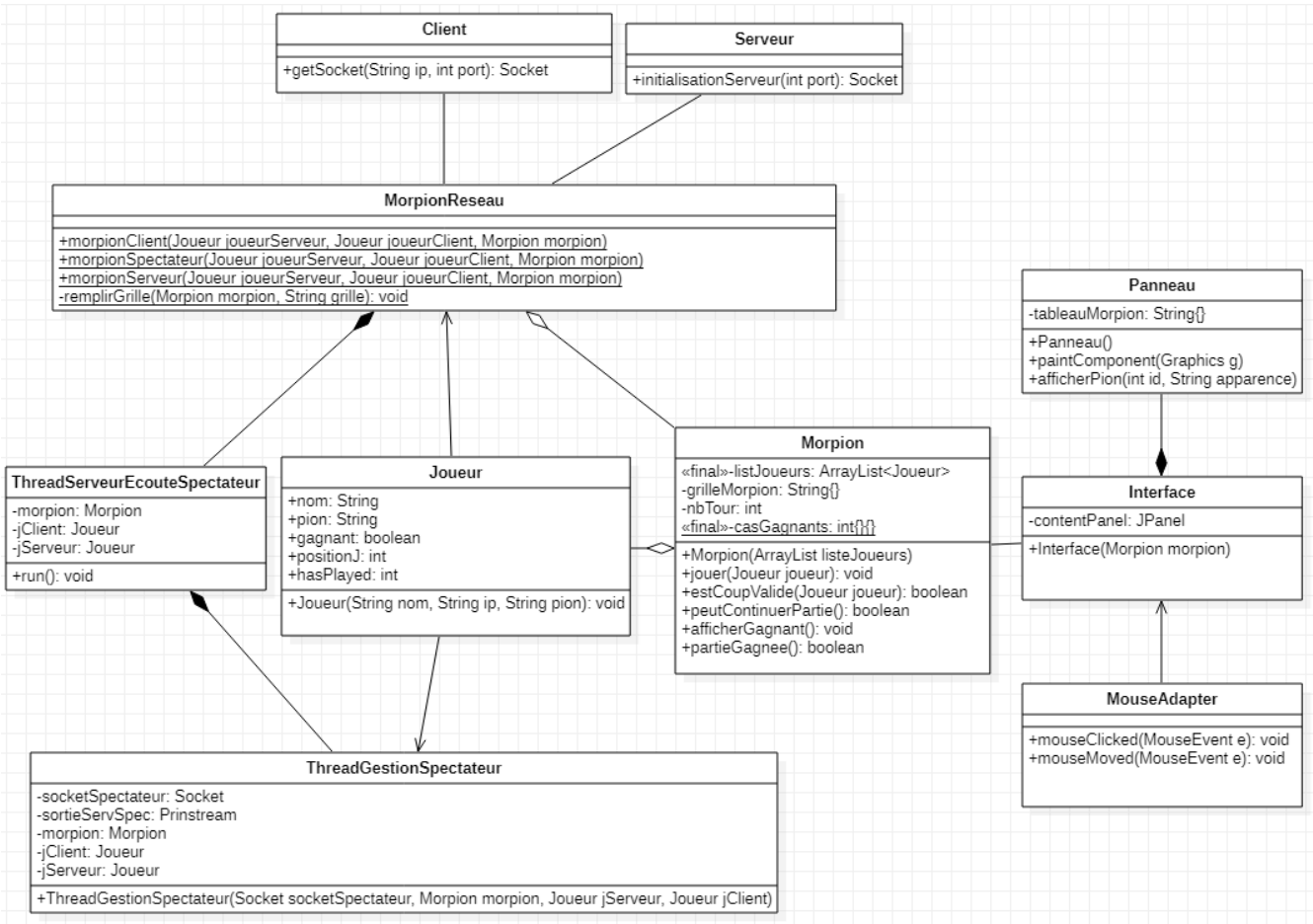


FIGURE 2.1 – Diagramme de classe

Ci-dessus, dans la figure 2.1 vous pouvez voir la structure de notre projet avec un diagramme de classe simplifié. On observe que la plupart des classes sont liées à MorpionReseau. En effet, MorpionReseau possède un attribut de type Morpion, car il gère le fonctionnement du morpion de l'utilisateur. Pour gérer le spectateur, MorpionResau possède aussi un attribut de type ThreadServeurEcouteSpectateur. Le Morpion possède 2 joueurs dans une liste, car il va les gérer pour notamment faire le lien entre le serveur et le client. La classe ThreadServeurEcouteSpectateur crée des objets de type ThreadGestionSpectateurs afin de gérer chaque spectateur.

2.1.1 Les Sockets

Pour notre projet nous avons utilisé la technologie socket*. Cette technologie permet de créer une connexion à travers la technologie TCP. Le protocole TCP est omniprésent dans notre environnement numérique, c'est pourquoi aborder son utilisation est capital. Les sockets sont très souvent utilisés dès

qu'il est nécessaire de procéder à une connexion entre plusieurs machines. La classe Socket de la bibliothèque Java "Closeable" était donc particulièrement utile afin d'implémenter cette technologie qui est fondamentale à notre projet. L'intérêt se portant sur la connexion entre deux machines, il est donc directement question de l'utilisation de cette technologie.

2.1.2 Les Threads

Malgré le fait que les sockets soient primordiaux pour effectuer une connexion entre 2 machines, ils présentent des limites lors de la connexion de multiples machines. Pour notre projet, nous devons permettre à plusieurs spectateurs de se connecter sur le même port d'un même serveur. Malheureusement, pour manipuler efficacement et facilement plusieurs spectateurs, nous devons pouvoir gérer indépendamment la multitude de connexions. Une des manières la plus élégante et efficace est d'utiliser des threads. En effet, les threads permettent de traiter indépendamment plusieurs fonctions, de façon à ce que l'utilisateur ait l'impression que les tâches s'effectuent en parallèle. Vous pouvez examiner l'illustration du fonctionnement de plusieurs threads ci-dessous dans la figure 2.2.

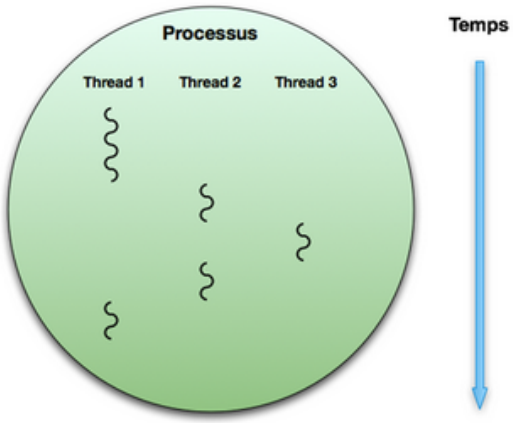


FIGURE 2.2 – Schéma explicatif thread

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 Morpion

L'objectif principal du projet comme dit précédemment était la gestion une communication réseau entre plusieurs ordinateurs. Pour ce faire, nous avons décidé de réaliser un morpion en réseau. Le choix d'un jeu simple permet de plus facilement nous concentrer sur la réalisation d'une communication réseau, tout en permettant d'appliquer de façon concrète et ludique cette technologie.

La classe Morpion est présentée ci-dessous.

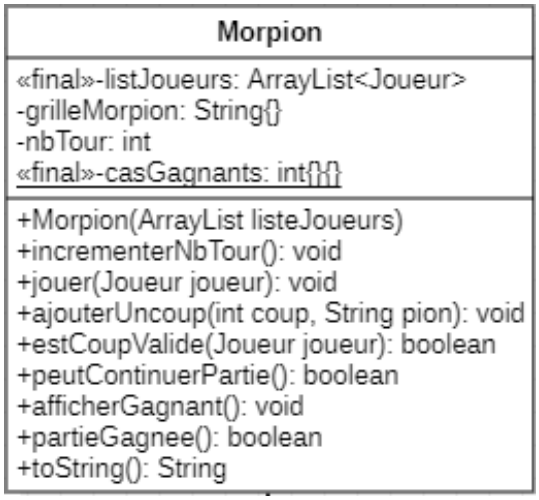


Figure 2.3 – Classe Morpion

Nous allons commencer par décrire les attributs. Nous avons une liste de joueurs qui va contenir les deux joueurs qui vont s'affronter ainsi qu'un tableau de String grilleMorpion, qui va permettre de stocker l'emplacement du pion des joueurs. Un entier nbTour qui, comme son nom l'indique, va stocker le nombre de tours. Enfin, afin de vérifier les cas gagnants, nous avons un tableau de int casGagnants qui regroupe toutes les configurations gagnantes. Maintenant que nous avons abordé les attributs, nous allons expliquer les différentes méthodes.

La méthode jouer prend en argument un joueur et attend la position du pion qu'il a choisie. Puis, avec la fonction estCoupValide nous vérifions que le coup du joueur respecte les règles, enfin, nous l'ajoutons à la grille. La fonction estCoupValide va vérifier si le coup est possible, c'est-à-dire s'il est situé dans la grille, mais également si l'emplacement choisi ne contient pas déjà un pion. La fonction peutContinuerPartie va retourner un boolean si la partie est finie. Par exemple, si le nombre de tours vaut 9 ou si partieGagnee retourne true, alors la partie est terminée.

La fonction partieGagnee vérifie s'il y a un gagnant. Par exemple, si le nombre de tours est inférieur à trois, alors il ne peut pas y avoir de gagnant, elle retourne donc false. Sinon, la fonction parcourt le

tableau grilleMorpion et le tableau casGagnants pour voir si un des joueurs est dans la configuration des cas gagnants, c'est-à-dire si 3 de ses pions sont alignés, si c'est le cas elle retourne true.

Le toString prend cette forme :

```
 Ton profil : Client X
 Adversaire : Serveur O
| | |X|
| |O| |
|X| | |
```

FIGURE 2.4 – Résultat du toString

2.2.2 Partie Réseau

Dans la partie purement réseau nous avons deux classes. Une classe client et une classe serveur.

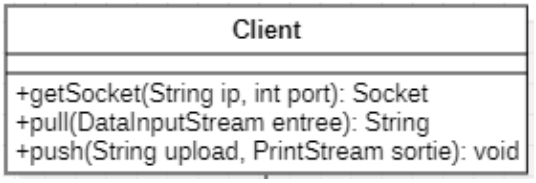


FIGURE 2.5 – Classe client

La classe client n'est composée que de méthodes static. Elle permet de récupérer un socket avec une adresse donnée, mais également d'envoyer et recevoir des données grâce aux fonctions push et pull.

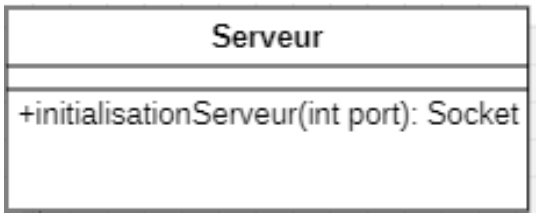


FIGURE 2.6 – Classe serveur

La classe serveur permet de créer un socket et d'écouter sur celui-ci, nous nous servons des méthodes de client.

2.2.3 Gestion Morpion en réseau

Client

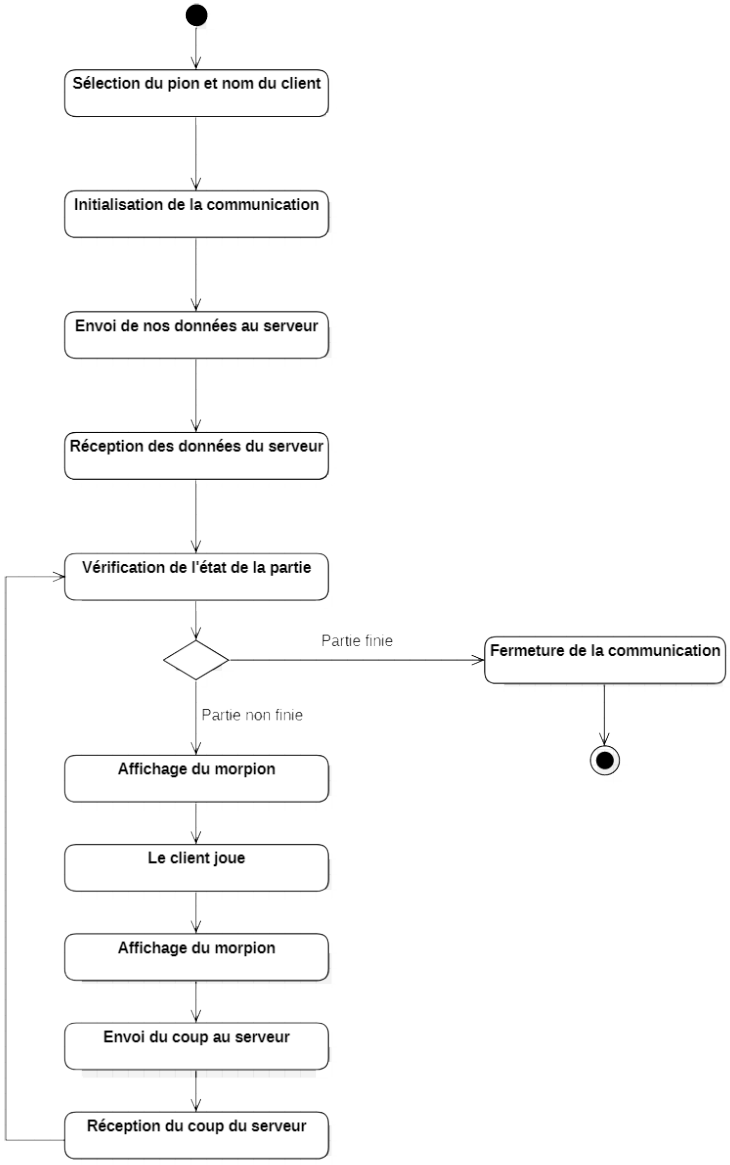


FIGURE 2.7 – Diagramme d'activité du client

Ci-dessus en figure 2.7, vous trouverez le fonctionnement du client. Le client est une méthode morpionClient dans notre classe MorpionRéseau. Le client rentre ses informations avec son pseudo et son pion, ensuite il se connecte au serveur et lui envoie ces mêmes informations. Le fait qu'il envoie ses informations permet au serveur d'identifier le client pour le reste de la partie. Une fois ces informations

transmises, le morpion est affiché et le client commence à jouer. Après cela, le morpion est réaffiché, et nous envoyons au serveur le coup que le client vient de jouer. Cette alternance d'affichage est nécessaire afin de mettre à jour l'affichage à chaque fois qu'il y a un nouveau coup, que ce soit de la part du client, ou reçu par le biais du serveur. Une fois que le client a envoyé le coup, nous attendons de recevoir celui du serveur, et nous rebouclons tant que la partie n'est pas finie. La fonction qui détermine si la partie est finie, permet de faire la différence entre une égalité ou un match gagné par un des joueurs, elle permet donc de manière sûre de savoir si le jeu peut ou non continuer et permet notamment d'éviter d'attendre un coup de la part du serveur, alors qu'il s'est déconnecté et a fini de jouer.

Le Spectateur

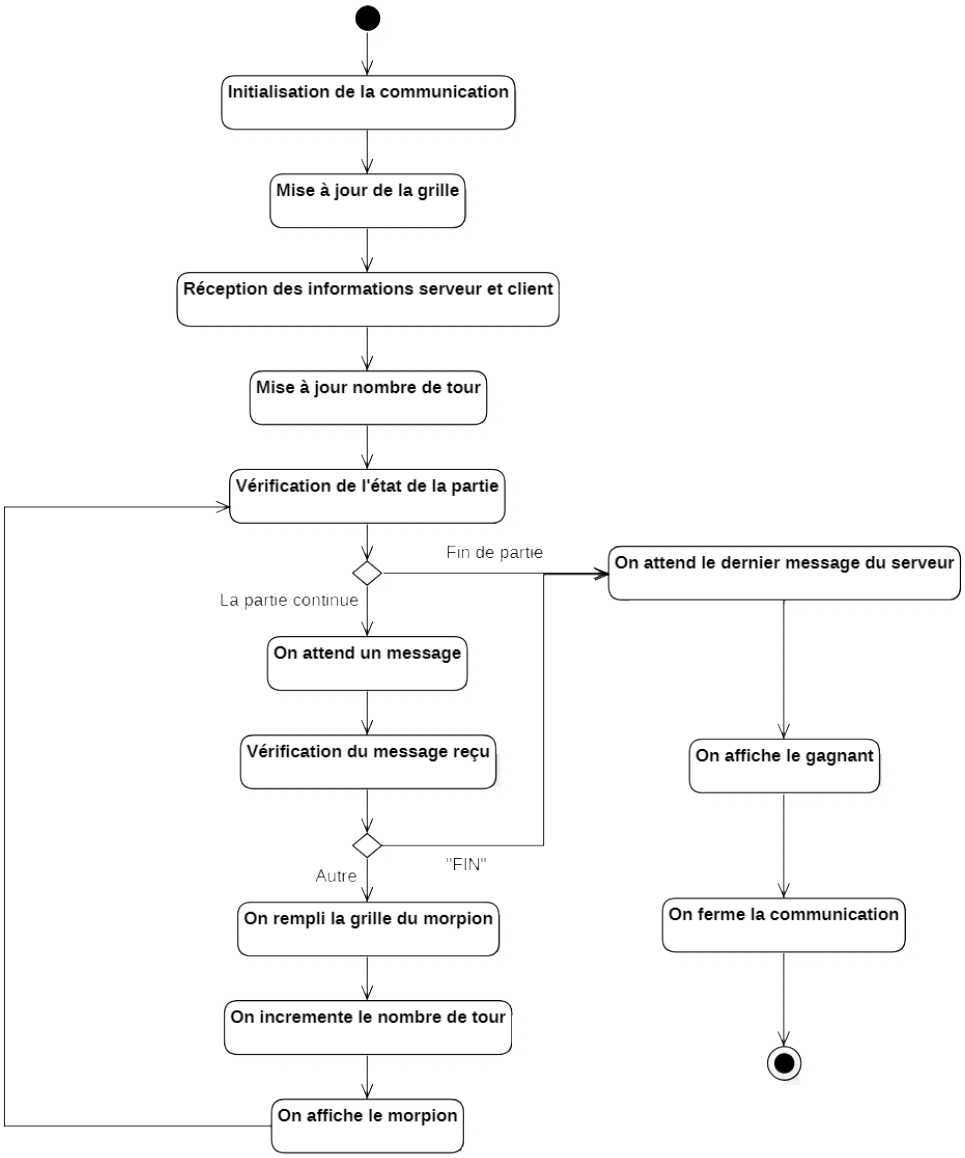


Figure 2.8 – Diagramme d'activité du spectateur

Le spectateur n'a pas besoin de pseudo dans notre programme, car nous avons décidé que les spectateurs ne seront pas visibles des joueurs. Ensuite nous recevons la grille, les infos des 2 joueurs et le nombre de tours. À partir de ce moment-là, nous rentrons dans une boucle similaire au joueur client, c'est-à-dire qu'on met à jour le morpion à chaque coup. À la place d'enregistrer seulement le coup, qui nécessite de savoir qui a joué, nous mettons à jour la totalité de la grille, par souci de simplicité. Avant de mettre à jour la grille, nous vérifions toujours si le message vaut "FIN", ce qui signifie que le serveur a mis fin à la partie, cela permet d'éviter toute erreur due au fait que le serveur n'est plus disponible, alors que le spectateur attend la suite de la partie. Si la partie est finie, alors nous attendons un dernier message du serveur, puis nous affichons le gagnant et nous clôturons la communication. Tous ces messages que nous attendons du serveur permettent de rester synchronisés avec celui-ci et évitent les erreurs dues à une déconnexion trop abrupte.

Le Serveur

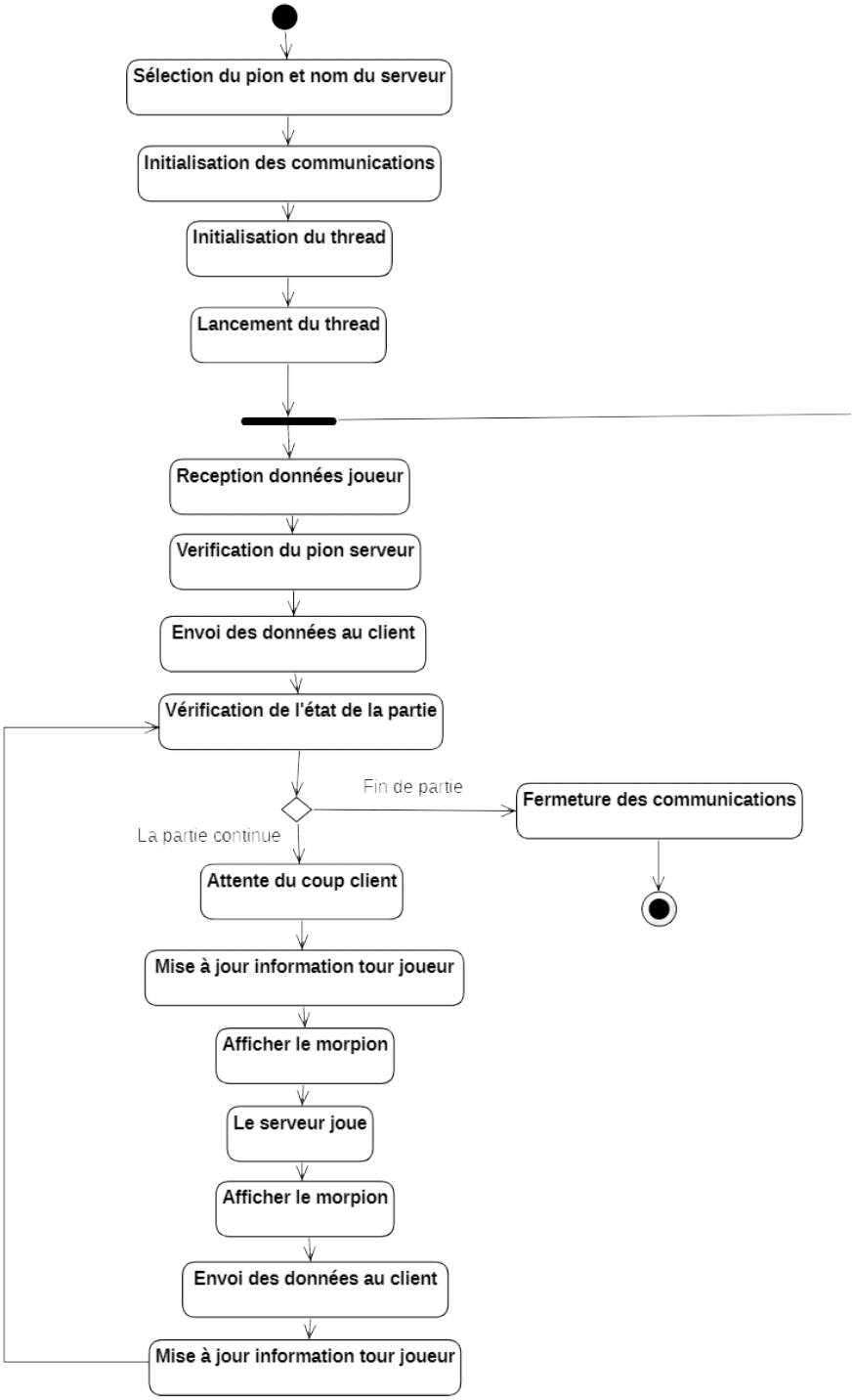


FIGURE 2.9 – Diagramme d'activité du serveur principal

Le début de la méthode serveur est très similaire à celle du client : l'utilisateur saisit ses infos, puis nous initialisons la communication avec le client et le spectateur. Le spectateur nécessite de lancer un thread, que nous détaillerons plus tard. La suite est pratiquement miroir de la partie client. En effet, il

commence d'abord par attendre le coup du client, et là il met à jour le fait que le client a joué, c'est utile pour le thread spectateur. Puis la méthode alterne entre envoi du coup, affichage du morpion et mise à jour du tour des joueurs, jusqu'à que la partie soit finie.

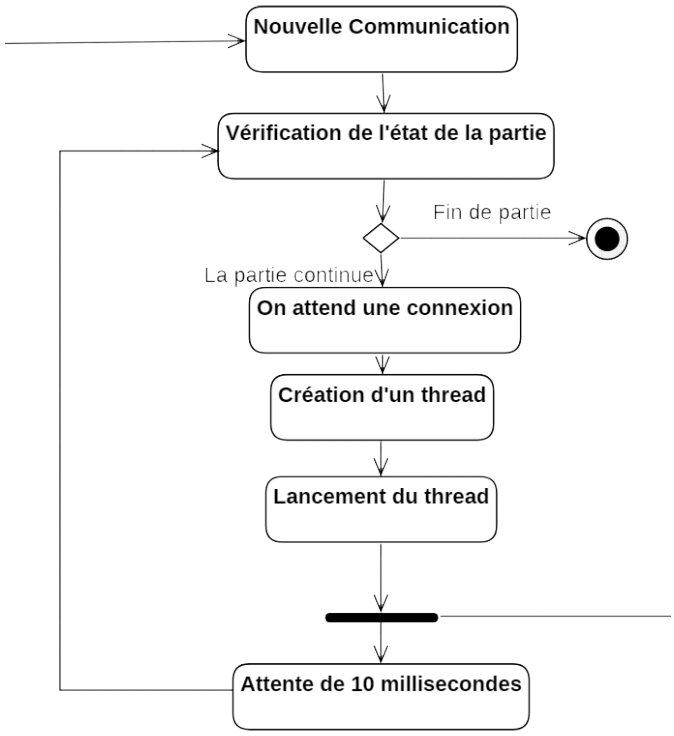


FIGURE 2.10 – Diagramme d'activité du thread d'écoute

Maintenant, examinons le diagramme d'activité du thread d'écoute du spectateur. Nous observons que le serveur d'écoute attend un nouveau spectateur. Quand un spectateur est connecté, le programme lance un nouveau thread de gestion du spectateur, tant que la partie n'est pas finie.

```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 du thread, un socket est initialisé au port 2001. Ce port est dédié aux spectateurs. Nous nous apercevons que nous restons dans la boucle While, tant que le morpion n'est pas fini, c'est-à-dire que morpion.peutContinuerPartie() est vraie. Ensuite, nous accédons à la fonction bloquante accept(), qui va permettre de créer un socket spectateur, dès qu'un utilisateur se connecte à ce port. Après la création de ce socket, nous créons directement un nouveau Thread contenant ce socket, le morpion et les joueurs. Le thread créé est immédiatement lancé. Ce Thread va se charger de la gestion du spectateur, c'est ce que nous détaillerons dans le schéma suivant. Ce code met en évidence la puissance des threads, qui permettent, en théorie, dans la limite des capacités de la machine, de connecter une infinité de spectateurs.

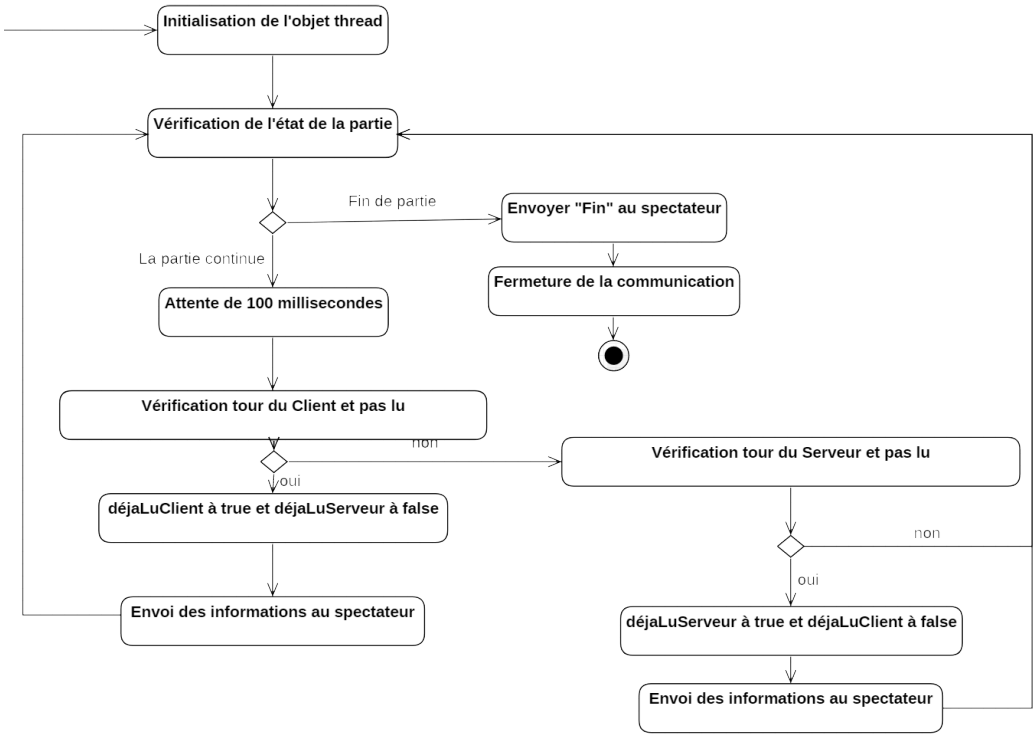


FIGURE 2.12 – Diagramme d'activité du thread gestion spectateur

Cette partie est le thread de gestion du spectateur, à chaque nouveau tour, elle envoie la grille du morpion au spectateur. La méthode attend que le client ou le serveur ait joué, et envoie le morpion à chaque fois qu'un des deux joueurs a joué.

2.2.4 Interface Graphique

Afin de rendre l'expérience de l'utilisateur plus agréable, nous avons décidé de mettre en place une interface graphique pour jouer au morpion.

Cette interface est donc utilisable une fois que la connexion a été établie entre les deux joueurs. Elle se résume à une fenêtre affichant une grille avec des cases numérotées. Chaque case correspond donc à

une case de morpion sur laquelle il est possible de jouer.

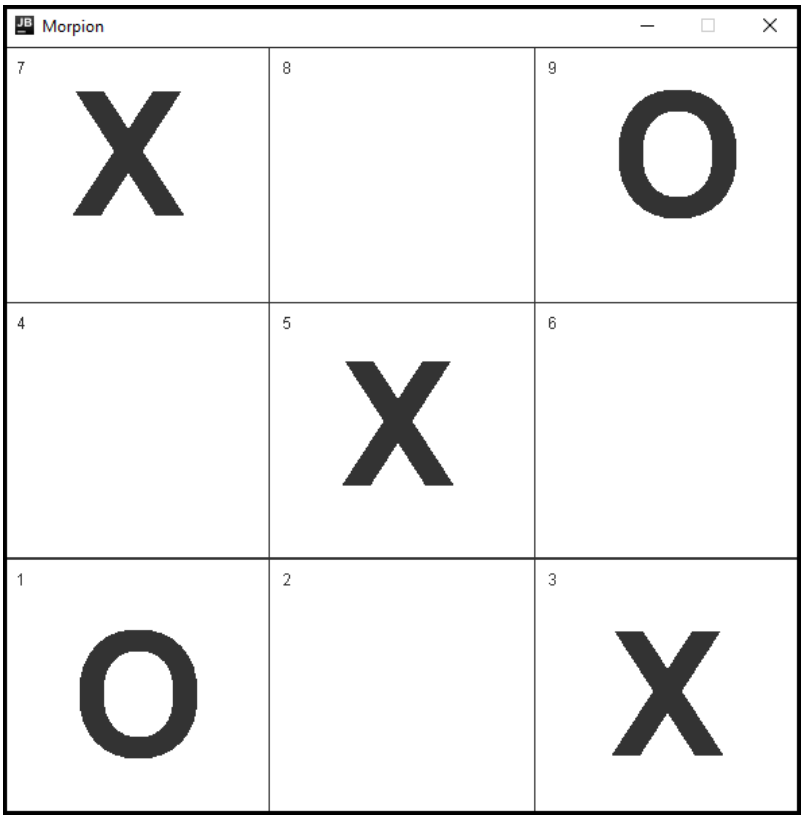


FIGURE 2.13 – Affichage du morpion dans l'interface graphique

Cliquer sur une case de la fenêtre indique donc à l'objet Morpion sur quelle case nous souhaitons jouer. L'interface envoie donc l'information au code du morpion de la même manière qu'avec l'utilisation du pavé numérique. Le code affiche alors l'état de la partie dans le terminal.

2.2.5 Arborescence

Pour pouvoir coder de manière efficace, nous avons décidé de structurer notre projet en trois différents packages. Une première partie comprend les classes nécessaires à l'interface graphique. Une seconde englobe toutes les classes utiles pour le fonctionnement du morpion en lui-même. La troisième partie comporte les classes contenant les méthodes utiles à la connexion client-serveur. La structure est donc observable dans la capture d'écran ci-dessous.

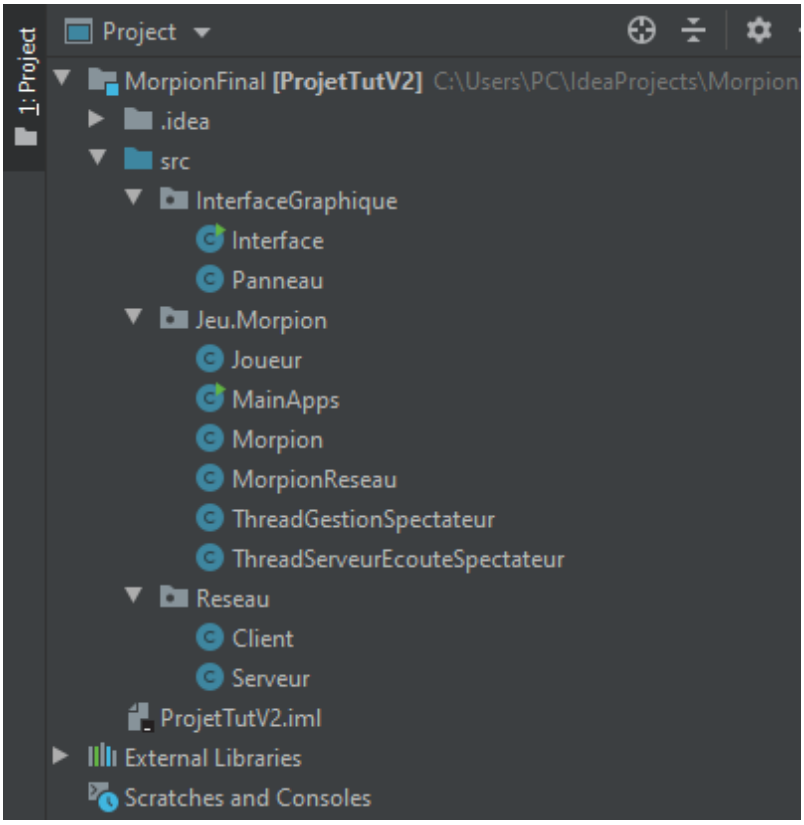


FIGURE 2.14 – Arborescence du projet

2.2.6 Validation, résultats et perspectives

Pour tester notre programme, nous avons exploité le débugger qui est intégré à IntelliJ IDEA et qui permet de faire du pas-à-pas dans le programme. Pour tester la partie réseau, nous avons utilisé des affichages dans le terminal, ce qui nous permettait de connaître l'état des variables qui circulent dans le réseau afin de vérifier le bon déroulement des échanges. C'était une option plus facile à mettre en place notamment avec les threads qui modifient parfois les variables de façon difficilement prévisible. De plus, le debugger n'était pas vraiment adapté à la vérification de la connexion du réseau.

Résultats et perspectives Nous avons rempli la quasi-totalité du cahier des charges Voici les différentes fonctionnalités que nous avons réussies à réaliser :

- Nous avons un morpion fonctionnel.
- Ce morpion permet de jouer en ligne avec un autre utilisateur.
- Au moins un spectateur peut regarder la partie, l'application peut même gérer plusieurs spectateurs, nous n'avons pas testé le nombre maximal.
- Enfin, l'interface permet d'avoir un affichage et une ergonomie plus attrayants.

Parmi les éléments qui étaient demandés, nous n'avons pas eu la possibilité de développer ces 2 fonctionnalités :

L'utilisateur ne peut pas sélectionner un jeu parmi une liste de jeux disponibles.

L'utilisateur ne peut pas sélectionner une partie parmi une liste de toutes les parties en cours.

Comme tout projet, il existe des améliorations possibles afin d'enrichir notre application au-delà de ce qui était demandé, certaines sont des détails, d'autres peuvent demander une restructuration plus importante du projet.

Le reste de l'affichage (nombre de tours, demande de saisie…) pourrait être aussi affiché sur l'interface graphique, la grille devrait être mise à jour au bon moment, et non pas quand nous déplaçons la souris.

Enfin, nous pourrions refactoriser le projet afin qu'il puisse être adapté à d'autres jeux, notamment en créant une interface Jeu, qui permettrait de rendre celui-ci très évolutif, en facilitant l'ajout de nouveaux jeux.

3. Résultats

3.1 Manuel d'installation

Pour récupérer le projet, nous devons passer par GitHub et copier le lien du projet dans IntelliJ.

Pour l'utiliser il vous faudra installer IntelliJ IDEA et exécuter la classe MainApps. Si jamais vous souhaitez jouer avec des personnes en local il vous faudra saisir l'adresse de celui qui compte héberger.

Pour jouer en multi-joueurs, il vous faudra une personne pour héberger, celle-ci devra ouvrir les ports du routeur, ce qui permettra aux clients qui ont la bonne adresse IP et le bon port de pouvoir se connecter à distance.

Pour connaître la démarche pour gérer les connexions au routeur, vous trouverez les informations à ce lien : https ://www.planete-domotique.com/blog/2018/05/30/la-redirection-de-port-cest-quoi/

3.2 Manuel d'utilisation

3.2.1 Démarrer le programme

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

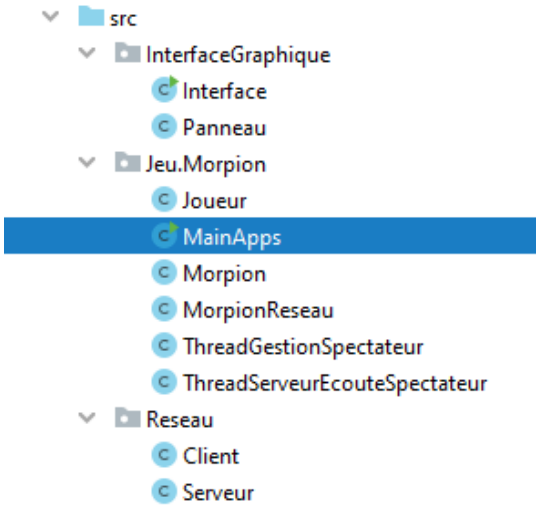


FIGURE 3.1 – Lancer l'application

Quand vous avez ouvert le projet avec IntelliJ IDEA, allez dans la classe MainApps visible ci-dessus.

Pour lancer le programme, cliquez sur le triangle vert, cela va compiler le programme.

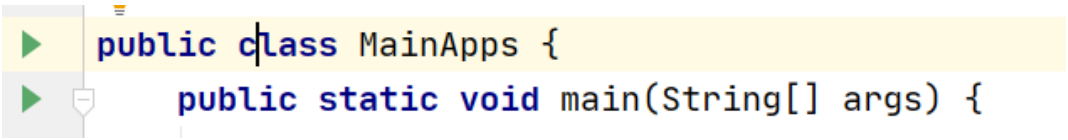


FIGURE 3.2 – Tutoriel compilation

3.2.2 Choix du mode de jeu

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

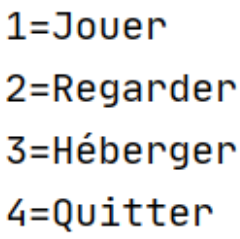


FIGURE 3.3 – Choix du mode de jeu

- Si vous voulez héberger une partie pour jouer avec un autre joueur, entrez "3" dans le terminal, puis validez avec "entrée".
- Si vous voulez rejoindre quelqu'un qui héberge déjà une partie, entrez "1",

— Si vous voulez regarder une partie entre 2 joueurs, entrez “2”.

— Pour quitter la partie, entrez 4, le programme s’arrêtera.

3.2.3 Mode Joueur

Saisie des informations

Si vous choisissez de jouer ou d’héberger, vous aurez une expérience très similaire. Le programme vous demandera d’entrer votre nom et votre pion :

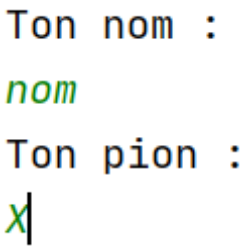


Figure 3.4 – Choix du nom et pion

Par défaut, le nom et pion de l’hébergeur sont “Serveur” et “O”, le client comme “Client” et comme pion “X”. N’entrez pas de pion avec un espace ou avec plusieurs caractères, le programme n’acceptera pas ces données et vous demandera de les insérer à nouveau.

Jouer une partie

Si vous ne voyez pas la fenêtre morpion présente ci-dessous, il faut la mettre en premier plan, car c’est la fenêtre où vous allez jouer.

Pour jouer, vous devez attendre le coup du client si vous êtes le serveur, sinon, cliquez sur la case de votre choix, afin d’y placer votre pion.

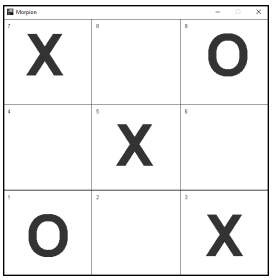


Figure 3.5 – Rendu morpion

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

3.2.4 Mode Spectateur

Si vous avez choisi le mode spectateur, alors vous pouvez visionner directement la partie sur le terminal, ou bien l'interface. En cas de problème d'affichage, positionnez la souris sur la fenêtre.

3.2.5 Fin de la partie

En cas de fin de la partie, c'est à dire dans le cas où un joueur a trois pions alignés, ou alors s'il ne reste plus de case vide, la partie se finit et vous pouvez examiner l'état de celle-ci sur le terminal, comme ci-dessous.

```
Tour n°5
 Ton profil : Client X
 Adversaire : Serveur O
|X| |O|
| |X| |
|O| |X|

Fin de la partie
Le gagnant est : Client X
```

FIGURE 3.6 – Fin de la partie

4. Gestion de projet

4.1 Démarche personnelle

Nous avons choisi de travailler par itération afin de construire au fur et à mesure le projet, et d'ajuster au besoin certains aspects que nous n'aurions pas prévus. Cette méthode nous permet, une fois que nous avons passé les premières étapes du projet, d'obtenir un travail que l'on peut présenter à tout moment, car nous ne faisons qu'ajouter des fonctionnalités à un programme déjà opérationnel.

Notre plateforme de communication est Discord, car elle est pratique pour communiquer efficacement par vocal, mais aussi pour partager l'écran. Ce sont des fonctions essentielles, surtout quand il nous est devenu impossible de nous rencontrer pendant la pandémie.

4.2 Planification des tâches

Afin de mener à bien notre projet, nous avions quatre étapes importantes : la première étant la réalisation d'une communication simple en réseau entre 2 terminaux, nous avions ensuite la conception du morpion en multi-joueurs locaux, la possibilité de jouer en réseau, puis finalement la possibilité d'avoir plusieurs spectateurs qui puissent à tout moment assister à la partie. Cela correspond plus ou moins à nos "sprints". Le projet a donc débuté le 23 janvier et la date de rendu du rapport ainsi que la soutenance se dérouleront début juin. Certaines tâches dépendant de la réalisation de précédentes étapes, voici donc le diagramme de Gantt prévu pour le projet :

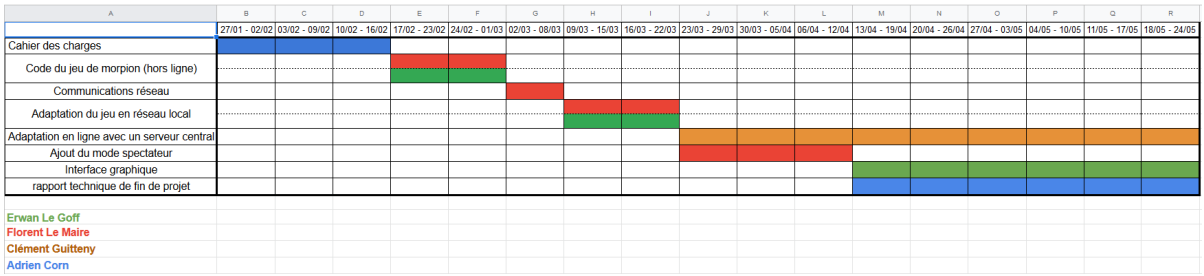


Figure 4.1 – Diagramme de GANTT

Le code du morpion et l'établissement des premières connexions réseau sont indispensables pour le développement de la suite du projet. En effet, avoir un jeu fonctionnel permet de plus facilement intégrer les fonctionnalités de communication. Sans morpion, notre projet aurait été beaucoup moins attractif à présenter. C'est pourquoi cela a constitué notre premier sprint. C'est seulement lorsque le jeu était globalement fonctionnel que nous avons étudié la gestion des communications en réseau, notamment à travers un simple chat, qui nous a permis de nous concentrer sur l'essentiel.

La réalisation du chat, afin de s'initier aux bases des sockets, constituait notre second sprint. Les cours de Francis Garcia et d'OpenClassroom nous ont permis de nous familiariser rapidement avec la notion des sockets, ceux-ci qui étant assez proches des tubes que nous avions vus en cours de réseau pendant cette période.

Une fois que notre chat était fonctionnel, la partie la plus importante et technique, était la mise en réseau du Morpion, c'était donc notre troisième sprint. Il a fallu organiser le code pour qu'il puisse facilement s'adapter au réseau. Heureusement, notre Morpion de base était bien réalisé, ce qui nous a permis de travailler à partir de bonnes bases. Avec le temps, nous avons pu observer nos progrès, notamment en programmation orientée objet, qui était encore un sujet flou au début de l'année, mais que nous avons vu en profondeur avec M. Palicov. Cela nous a permis de revenir sur certains codes afin de les optimiser, grâce à l'expérience que nous avions acquise.
Une fois que nous avions terminé cette fonction clé du projet, nous avons enchainé sur les 4 derniers sprints : un sprint parallèle, consistant à la réalisation d'une interface graphique. Quant à la partie réseau, un second sprint étant l'implémentation d'un spectateur unique qui doit obligatoirement rejoindre le jeu dès le début de la partie. Cette simplification a permis de mettre le doigt sur la complexité de la gestion d'un troisième appareil, mais aussi sur l'utilité des threads pour gérer plusieurs spectateurs arrivant en cours de partie. C'est pour cela que le sprint suivant consistait à rendre possible la connexion d'un spectateur en cours de partie, grâce à un thread. Puis finalement, le dernier sprint consistait à faire en sorte que plusieurs spectateurs puissent visualiser la partie en se connectant à tout moment. Nous avons globalement bien organisé notre travail. En effet, notre projet était présentable rapidement pour la soutenance. Nous n'avons fait qu'ajouter de la complexité et de la matière, afin de le rendre de plus en plus attractif.

4.3 Bilan critique par rapport au cahier des charges

À l'issue de ce projet, nous avons réussi à remplir toutes nos attentes par rapport au cahier des charges que nous nous étions fixé. En revanche, bien que notre projet soit entièrement opérationnel notre répartition des missions ne correspond pas parfaitement à celle indiquée dans le diagramme de Gantt de notre cahier des charges. En effet, lorsqu'une tâche était conclue, ses responsables venaient prêter main-forte aux autres membres du projet sur les tâches prioritaires.

Conclusion

À l'issue de ce projet, nous avons réussi à produire un jeu de morpion en réseau fonctionnel. Ce programme est notamment doté d'une fonction permettant à plusieurs spectateurs de regarder la partie. De plus, une interface graphique permet de rendre l'utilisation du morpion plus attrayante.

Ce projet nous a ainsi permis de développer nos compétences en programmation Java, mais aussi nos connaissances sur l'utilisation des outils Thread et Socket dans le cadre d'une application en réseau. Nous avons également renforcé notre capacité de travail en équipe et notre aptitude à nous organiser et nous répartir les missions.

Le projet tutoré nous a également appris dans des situations concrètes à établir un cahier des charges, à réaliser des diagrammes UML, mais aussi à rédiger un rapport de projet.

Des évolutions sont désormais envisageables, notamment concernant l'amélioration de l'interface graphique, ainsi que la restructuration du code pour faciliter l'ajout de différents jeux compatibles.

Bibliographie

Le cours “Java et la programmation réseau” de OpenClassrooms pour comprendre et apprendre l’utilisation des sockets et le principe de serveur-client https ://openclassrooms.com/fr/courses/2654601-java-et-la-programmation-reseau

Les cours de Mr Francis Garcia en Architecture Réseaux dans le cadre de notre formation à l’IUT d’Informatique

Annexes techniques

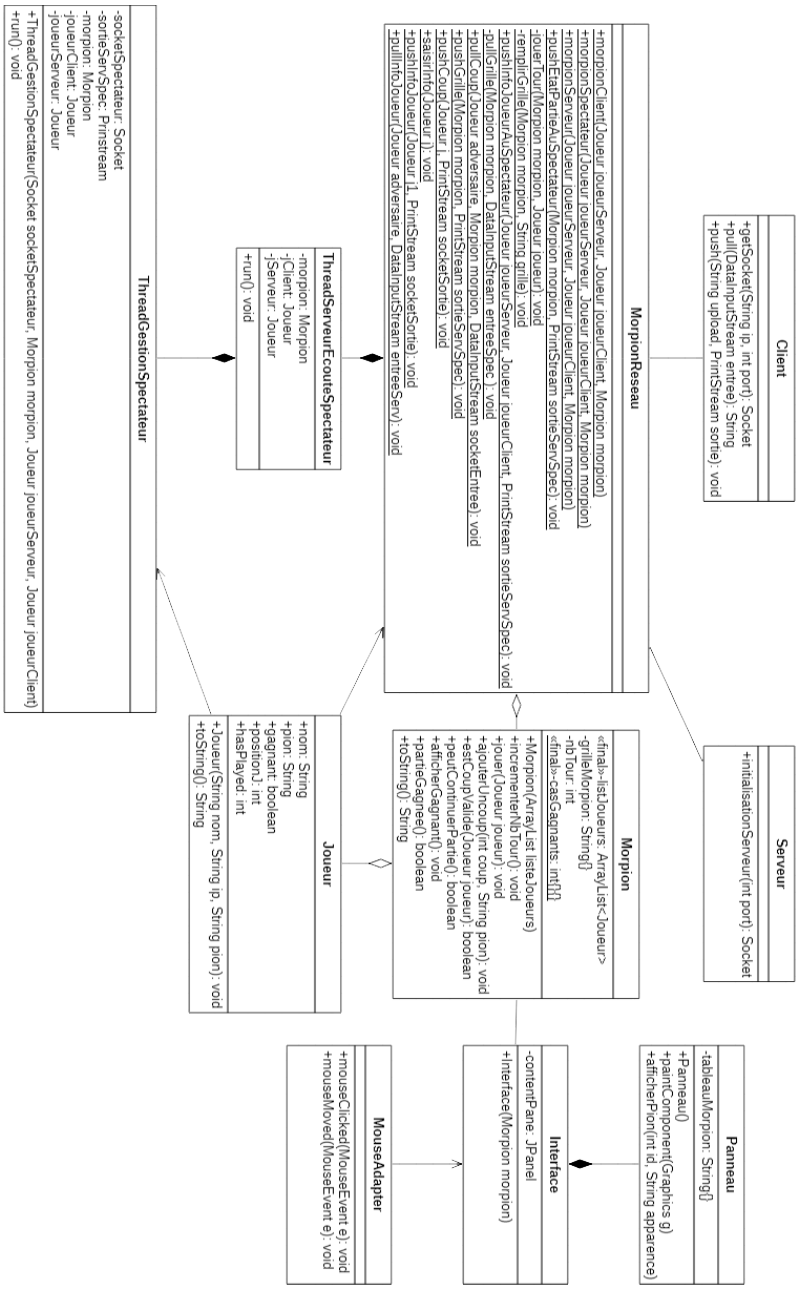


Figure 4.2 – Diagramme Classe complet

GRILLE DE RELECTURE DU RAPPORT

Cocher au moyen d'une croix chacun des items après vérification.

	Pour l'ensemble du rapport de projet	
	Mise en forme du propos	
	Police de caractère et de taille uniforme pour le corps du texte (ex : Times ou Calibri, 12 pt).	
	L'alignement est justifié	
	La présentation du texte est aérée : marges et interlignes raisonnables ; espace avant/après le paragraphe ; espace après les titres et sous-titres (Format > paragraphe)	
	Les titres	Les **titres des grandes parties** et différents avant-textes et post-textes **figurent en haut des pages** (insertion de sauts de page afin d'éviter les décalages à chaque modification)
		Ils ne sont pas suivis d'un signe de ponctuation (« : » ou « . »)
	En-têtes et pieds de pages	L'**en-tête** comporte le nom du rapport et le nom des auteurs (sauf sur les avant-textes et les post-textes qui n'ont pas d'en-tête).
		L'en-tête commence à l'introduction et disparaît après la conclusion
		Les **pieds de page** comportent les numéros de pages sur celles devant être numérotées
	Formulation du propos (langue, expression)	
	L'intégralité du texte est passée au **correcteur orthographique**, a été relue par une tierce personne et a été corrigée. Le texte ne comporte plus aucune faute.	
	La **ponctuation** est utilisée de façon pertinente : articulation des idées, pauses, phrases de longueur raisonnable permettant une bonne compréhension des propos.	
	Fréquemment utilisés, les **connecteurs logiques** permettent la compréhension des propos en reliant les idées logiquement (cause, conséquence, exemples, but, etc.)	
	Les **prénoms et noms** des personnes sont indiqués dans l'ordre Prénom + NOM (en majuscules)	
	Monsieur est abrégé « M. » et non « Mr » comme en anglais	
	Les **chiffres** sont écrits en lettres (« dix heures », « vingt-cinq employés », sauf exception [chiffre suivi d'une unité de mesure par exemple : 10 Mo, 10 km/h…]	
	Relecture orthographique	Indiquez les mots dont l'orthographe est à vérifier pour le rapport [ex. : base de donnéeş, programmation multi-agentş…]
	Les avant-textes	
	Numérotation en chiffres romains [sauf la page de couverture]	
		Le **titre du rapport** indiqué sur la page de couverture décrit précisément le projet.
		Sans numérotation

1

FIGURE 4.3 – Grille d'évalutation 1

	Page de couverture	Type de document : rapport de projet
		Logos des institutions [université de Montpellier, IUT]
		Indiquer le diplôme préparé
		Auteurs, tuteurs/tutrices
		Année universitaire
	Page de garde	Page vide
	Remerciements	Citez les personnes qui vous ont aidés et soutenus.
	Sommaire	Fidèle au plan appliqué par la suite
		Pas plus de 3 niveaux de titre
		Il n'y a **pas de titres « orphelins »** dans le plan. Ex 1.1.1 [et pas de 1.1.2]
		Il est inséré automatiquement et a été mis à jour une dernière fois avant l'impression
		Il peut être cliquable dans sa version numérique [facultatif]
	Glossaire	Il comporte les termes découverts pendant le projet
		Pas de mots anglais pour lesquels un terme français existe
		Les mots du glossaire sont **identifiables dans le texte au moyen d'un astérisque** lors de la première occurrence.
	Table des figures	Elle est insérée automatiquement et a été mise à jour une dernière fois avant l'impression
		Toutes les figures sont présentes [numérotation + légende]
	Corps du document	
	Les pages sont numérotées en chiffres arabes et reprennent à 1 [avant-textes en chiffres romains non comptabilisés — utiliser des sections pour cela]	
	Le plan est cohérent et permet d'illustrer l'ensemble du projet	
	Insertion de notes de bas de page pour donner des informations complémentaires ou suggérer des lectures/sources d'informations supplémentaires.	
	Articulation, enchaînement : chaque nouvelle partie est précédée d'une transition [essentielle !]	
	Les titres	Les titres des parties sont numérotés sous la forme 1.1.1 et sont indentés
		Pas de titres orphelins [ex. : 1.1.1 mais pas de 1.1.2]
		Ils ne sont pas suivis d'un signe de ponctuation « : » ou « . »
	Les renvois dans le texte	
	Figures et diagrammes :	Ils sont présents car nécessaires à la compréhension des propos [et non pour remplir une page, illustrer des éléments non explicités, etc.]
		L'ensemble des figures et diagrammes sont numérotés et légendés [De quoi s'agit-il ? Qu'apporte cet élément supplémentaire ?] avec un titre précis, et apparaissent dans la table des figures.
		Chaque figure et/ou diagramme est commenté et il y a un renvoi

2

FIGURE 4.4 – Grille d'évalutation 2

		dans le texte indiquant au lecteur quand il doit la/le regarder [« cf. figure 1 »]. C'est une illustration du propos et non l'inverse.
		Les extraits/illustrations de codes doivent être lisibles [attention aux fonds noirs, aux caractères trop petits, etc.]
	Les annexes	Un renvoi est effectué dans le texte, indiquant au lecteur quand il doit consulter chacune d'entre elles.
	Citations [attention au plagiat, même involontaire !]	Toute partie inspirée d'un document consulté doit mentionner la source selon les normes IEEE (numéro de la référence entre crochet Ex. : [2]) et permettre de retrouver le document dans la bibliographie.
		Les citations insérées sans modification respectent les normes de présentation IEEE
Progression détaillée		
	Introduction	Présentation du projet [contexte, objectifs] et du plan du rapport
		L'annonce du plan correspond à celui effectivement suivi
	Analyse	Analyse du contexte, des besoins fonctionnels et non fonctionnels
		Toute personne extérieure au domaine de l'informatique est en mesure de comprendre ce qu'elle contient.
	Rapport technique	Justification des choix de conception et de développement
		Il faut écrire cette partie en s'adressant à des informaticiens.
	Résultat	En fonction de votre projet vous pouvez présenter les tests, un manuel d'utilisation et/ou d'installation.
	Rapport d'activité	Planification et organisation du travail ; recul sur le travail effectué
	Conclusion	Synthèse et bilan
Les post-textes		
	Numérotation en chiffres romains qui reprend à I	
	Bibliographie	Elle doit impérativement mentionner les informations demandées dans une bibliographie comme l'auteur.e, la date de parution, etc. Il ne peut en aucun cas s'agir d'une liste d'adresse URL.
		Présentée selon les normes IEEE [utilisation de Zotero vivement conseillée !]
	Annexes techniques	Les annexes sont présentées dans une table
		Toutes les annexes portent un titre et sont numérotées
		Un renvoi est effectué dans le texte, indiquant au lecteur quand il doit regarder chacune des annexes.
	La quatrième de couverture	Résumé rédigé en français ET en anglais, qui reprend de façon fiable le contenu du rapport. Il comporte environ une centaine de mots.
		Mots-clefs relatifs au projet en français et en anglais

3

FIGURE 4.5 – Grille d'évalutation 3

Diagrammes techniques (uniquement si nécessaire)		
Section	**Objectif**	**Diagrammes UML**
Analyse des besoins	Objectifs de la section : - Identifier clairement le contexte dans lequel le logiciel à produire va s'insérer - Identifier les besoins des utilisateurs	Diagramme des packages Diagramme de cas d'utilisation Diagramme d'activité
Conception	L'objectif de cette section est de donner une vue logique du logiciel à produire. Nous devons identifier toutes les entités du domaine; les processus qui vont agir sur ces entités; et les règles d'interaction (règles métier)	Diagramme de classes Diagramme d'objets Diagramme de séquence Diagramme d'activités Diagramme de collaboration Diagramme d'état-transition
Réalisation	L'objectif de cette section est de présenter l'architecture interne du logiciel réalisé en termes de composants ainsi l'architecture de déploiement.	Diagramme de composants Diagramme de déploiement
Test et Validation	L'objectif de cette section est de montrer comment le logiciel réalisé a été validé et testé dans son environnement de fonctionnement.	Diagramme de déploiement

4

FIGURE 4.6 – Grille d'évalutation 4

Résumé : Ce projet tutoré nous a été confié dans le cadre de notre formation à l'IUT Informatique de Montpellier en année spéciale. Il consiste en la réalisation d'un jeu en réseau qui permet à deux joueurs d'interagir à distance, avec la possibilité d'accueillir plusieurs spectateurs au cours de la partie. Au travers de ce projet, nous avons pu utiliser de manière concrète des notions fondamentales et omniprésentes telles que les threads et les sockets. Ce rapport fait état du bon respect des contraintes fixées par le cahier des charges, mais aussi des différents axes d'améliorations envisagés.

Abstract : This tutored project was entrusted to us as part of our training at the « IUT Informatique de Montpellier » in a « special year ». It consists in the realisation of a network game that allows two players to interact remotely, with the possibility of hosting several spectators during the game. Through this project, we were able to concretely use fundamental and omnipresent notions such as threads and sockets. This report shows the good respect of the constraints set by the specifications, but also the different axes of improvement considered.