

29/04/2019

RAPPORT COO

Présenté par :

NITEKA Lys Ciella, JALAL Réda et NABIL Inès.

Projet de développement – Encadré par **Elena CABRIO**

SOMMAIRE

GLOSSAIRE	3
a) Présentation de l'architecture d'un système client/serveur.....	2
b) Diagramme de classes.....	2
c) Use-Case	2
d) Diagramme d'activité.....	2
INTRODUCTION	3
I. Diagramme d'activité	4
II. Diagramme de classe	6
a) Éléments du jeu	8
b) Création d'une partie	9
c) Lancement d'une partie	9
d) Les utilisateurs	10
e) Gestion des cartes.....	10
f) Gestion des merveilles	10
g) Le diagramme en détail.....	11
III. USE CASE.....	13
a) Use Case côté client	13
b) Use Case côté serveur	14
IV. Diagramme séquence	16
a) Diagramme de séquence du joueur (cf use case cote serveur)	16
b) Diagramme de séquence du serveur (cf use case côté client)	17
V. Interaction client serveur	19
CONCLUSION.....	20

GLOSSAIRE

a) Présentation de l'architecture d'un système client/serveur

L'environnement client/serveur désigne un mode de communication organisé par l'intermédiaire d'un réseau et d'un interface Web entre plusieurs ordinateurs. Cela signifie que des machines clientes (machines faisant partie du réseau et ayant donc l'adresse ip et le port du serveur) contactent un serveur, une machine généralement très puissante en termes de capacités d'entrées-sorties, qui leur fournit des services. Lesquels services sont exploités par des programmes, appelés programmes clients, s'exécutant sur les machines clientes.

b) Diagramme de classes

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci. Ce diagramme fait partie de la partie statique d'UML car il fait abstraction des aspects temporels et dynamiques.

Une classe décrit les responsabilités, le comportement et le type d'un ensemble d'objets. Les éléments de cet ensemble sont les instances de la classe.

c) Use-Case

Les diagrammes de cas d'utilisation sont généralement appelés diagrammes de comportement utilisés pour décrire un ensemble d'actions (cas d'utilisation) que certains systèmes (sujets) devraient ou peuvent exécuter en collaboration avec un ou plusieurs utilisateurs externes du système (acteurs).

Chaque cas d'utilisation devrait fournir des résultats observables et utiles aux acteurs ou autres parties prenantes du système.

d) Diagramme d'activité

Un diagramme d'activité fournit une vue du comportement d'un système en décrivant la séquence d'actions d'un processus. Les diagrammes d'activité sont similaires aux organigrammes de traitement de l'information, car ils montrent les flux entre les actions dans une activité.

Les diagrammes d'activité peuvent, cependant, aussi montrer les flux parallèles simultanés et les flux de remplacement.

INTRODUCTION

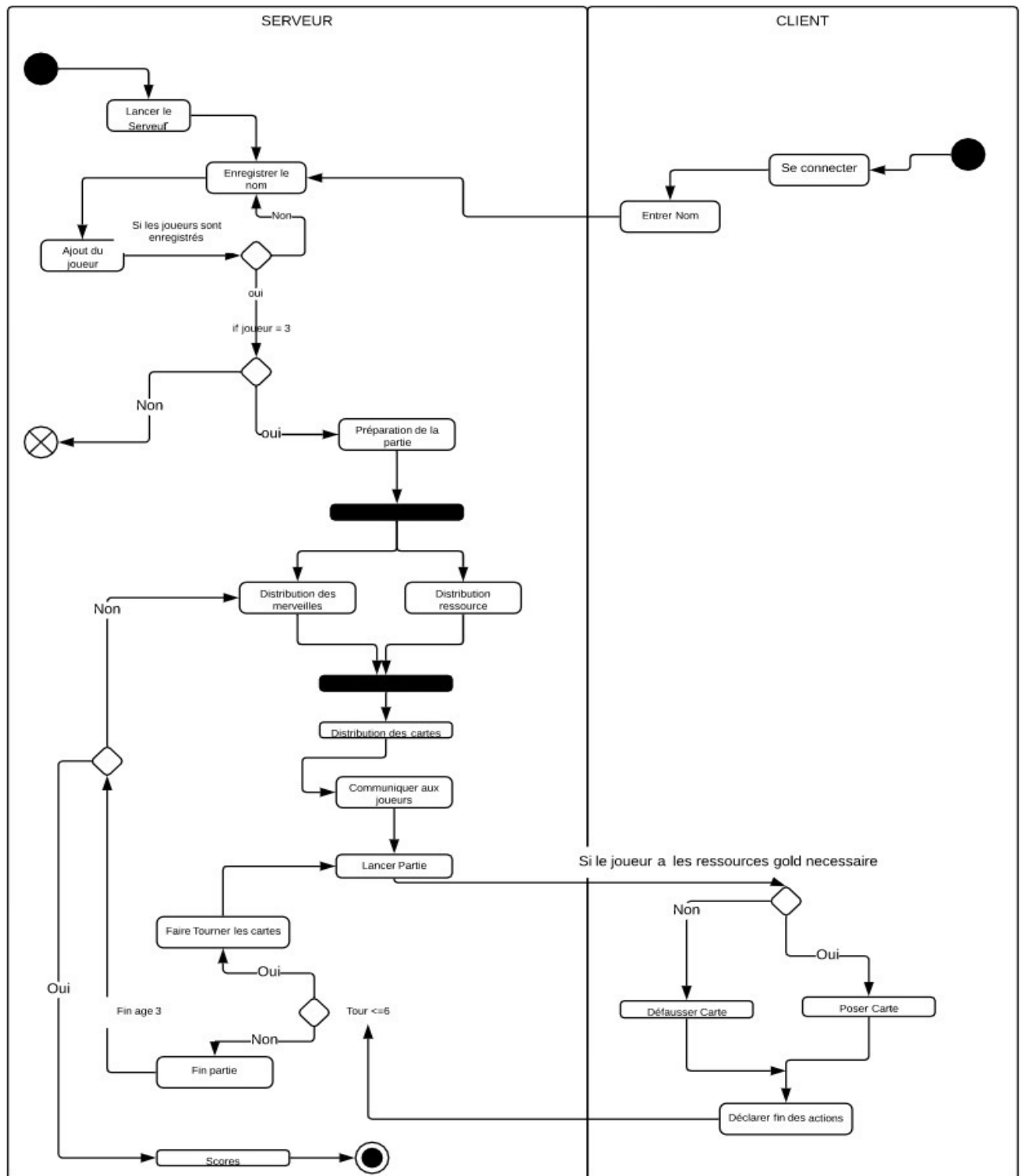
"Seven Wonders" est un jeu dont le thème est les sept merveilles du monde. C'est un jeu évolutif ayant pour but la construction des bâtiments des sept merveilles. Le jeu se joue jusqu'à sept joueurs. La main de départ contient sept cartes. Les joueurs développent leur civilisation autour de sept types de cartes.

Le principe est qu'à un tour, chaque joueur choisit une carte de sa main à jouer et passe le reste à son voisin. Ces derniers vont répéter l'action jusqu'à ce que 6 cartes des 7 en mains aient été jouées.

Nous avons alors pour but de modéliser le jeu de Seven Wonder et de le développer en java.

Nous allons exposer nos différents diagrammes de conception afin de montrer la façon dont nous allons procéder.

I. Diagramme d'activité



→ Tout d'abord, le serveur se lance le jeu puis les clients se connectent en donnant les noms qu'ils vont utiliser en tant que joueur.

→ S'il y'a 3 joueurs (sinon la partie n'est pas lancée), le serveur les enregistre en ajoutant chacun par la suite dans la partie

→ Une fois les joueurs ajoutés, le serveur commence la partie en distribuant les cartes (les cartes merveilles et les cartes ressources). Avant le positionnement des joueurs (gauche et à leur droite), le serveur lance la partie

→ Après le lancement de la partie, les joueurs choisissent la carte qu'ils désirent poser, ou défausser en fonction de leurs ressources et de leurs ors.

→ Le serveur va attendre de recevoir un signal lui communiquant la fin des actions des joueurs pour pouvoir choisir s'il enclenche un nouveau tour ou si l'âge est terminé (s'il y a eu plus de 6 tours).

→ Lorsque l'âge se termine, le serveur a le choix entre lancer l'âge suivant (si l'âge actuelle n'est pas 3) ou justement déclarer la fin de la partie à l'âge 3.

→ A la fin de la partie, il affiche les scores et termine ses actions.

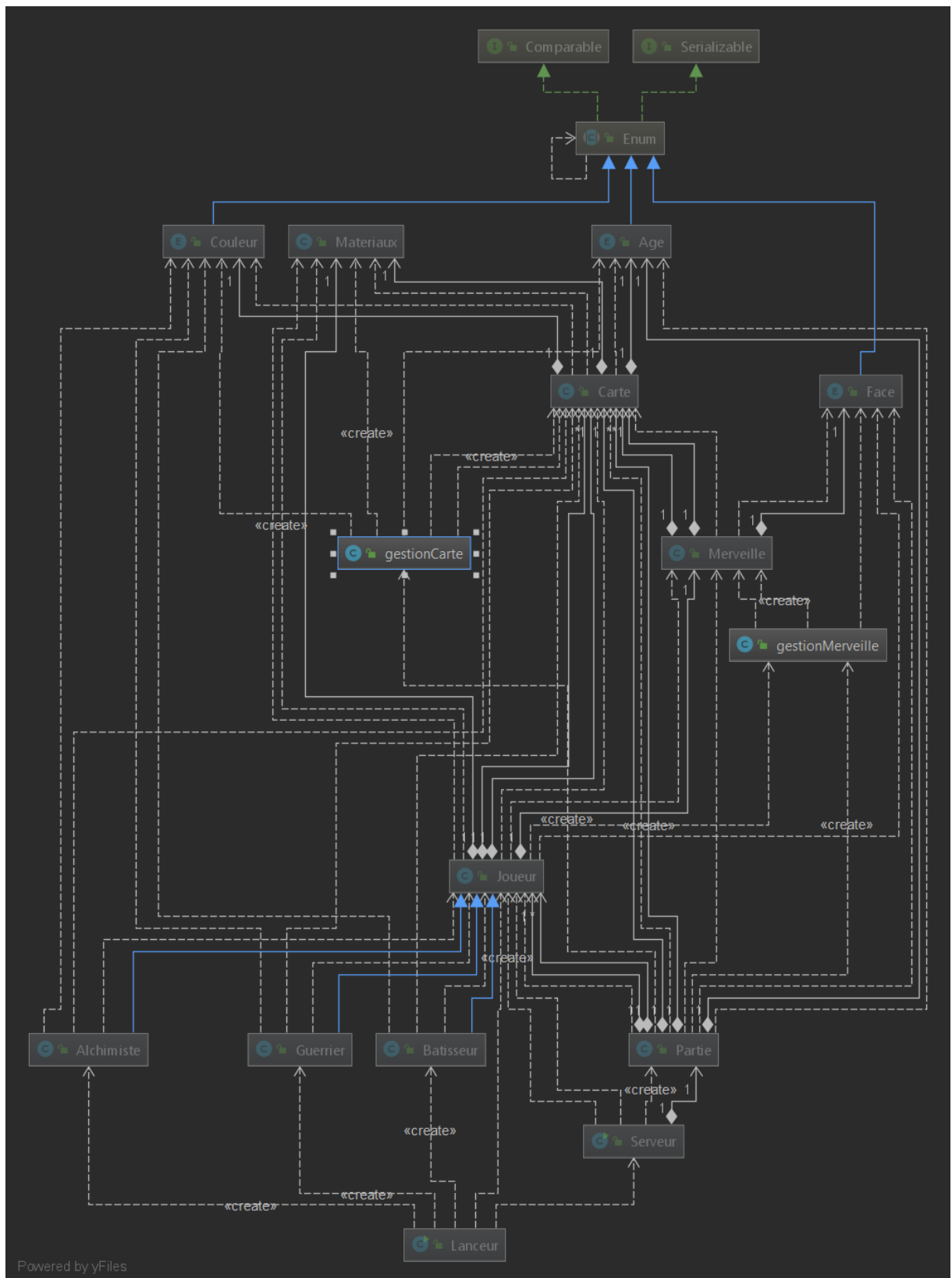
II. Diagramme de classe

Le diagramme de classe qui va suivre représente les classes et les interfaces du jeu que nous développons : Seven Wonders (voir figure 1).

Les éléments du diagramme sont :

- Une classe Carte : qui va attribuer aux cartes plusieurs caractéristiques dont la couleur, la face, le matériau, l'effet (il peut y en avoir plusieurs), l'âge correspondant.
- Trois énumérations qui vont correspondre aux caractéristiques précédentes, c'est-à-dire la couleur, la face, et l'âge.
- Une classe Merveille : elle va être caractérisée par un nom, un identifiant et une face attribuée.
- Une classe gestionMerveille : qui permet de gérer les Merveilles.
- Une classe gestionCarte : qui permet de gérer les cartes.
- Une classe Matériaux : qui va contenir les caractéristiques d'une carte.
- Une classe Partie : qui gère le déroulement d'une partie.
- Une classe Joueur : qui prend en compte le nom et l'url pour identifier un joueur après la connexion au serveur, ensuite elle déclare la réception des merveilles et des cartes aux joueurs.
- Une classe Serveur : qui va servir de relation aux différentes classes, et qui va permettre aux joueurs de s'identifier, permettre de lancer une partie et ainsi de suite avec la distribution des cartes.
- Une classe Lanceur : qui va permettre de générer une partie dès que le serveur l'aura signalé (quand les joueurs seront connectés).

Figure n°1 : Diagramme de classe en résumé.

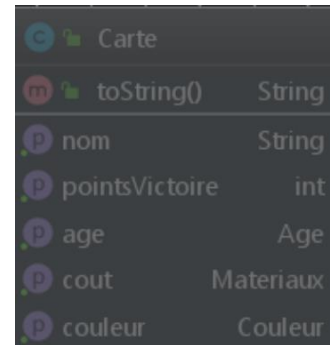


a) Éléments du jeu

La classe Carte :

Elle est constituée de plusieurs caractéristiques, les suivantes :

- Age (à partir de l'énumération Age)
- Couleur (à partir de l'énumération Couleur)
- Nom (String)
- Cout (Matériaux en Integer)
- Point de victoire (Integer)

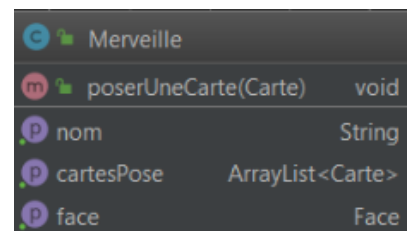


C	Carte	
m	toString()	String
p	nom	String
p	pointsVictoire	int
p	age	Age
p	cout	Matériaux
p	couleur	Couleur

La classe Merveille :

Elle est constituée de plusieurs caractéristiques, les suivantes :

- cartesPose (Array List<Carte>)
- Face (à partir de l'énumération Face)
- Nom (String)

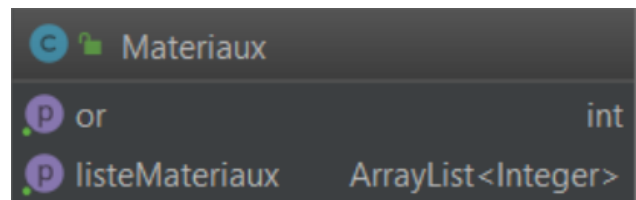


C	Merveille	
m	poserUneCarte(Carte)	void
p	nom	String
p	cartesPose	ArrayList<Carte>
p	face	Face

Elle est composée de la méthode « poserUneCarte(Carte) » : elle va permettre de prendre en paramètre une carte instanciée, et de l'ajouter à la liste des cartes posées (cartesPose Array List).

La classe Matériaux :

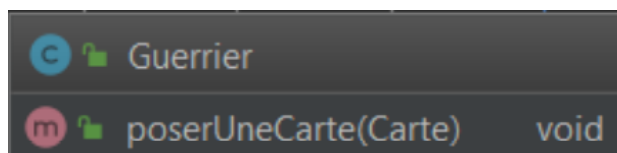
Cette classe contient les différents matériaux contenus dans le jeu, qui seront stockés dans une liste.



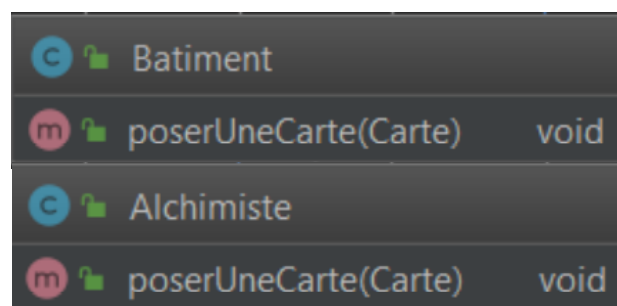
C	Matériaux	
p	or	int
p	listeMatériaux	ArrayList<Integer>

Les classes Guerrier, Batiment, Alchimiste :

Ce sont les différents types de profils des joueurs auxquels on aura assigné les cartes qui leurs correspondent.



C	Guerrier	
m	poserUneCarte(Carte)	void



C	Batiment	
m	poserUneCarte(Carte)	void
C	Alchimiste	
m	poserUneCarte(Carte)	void

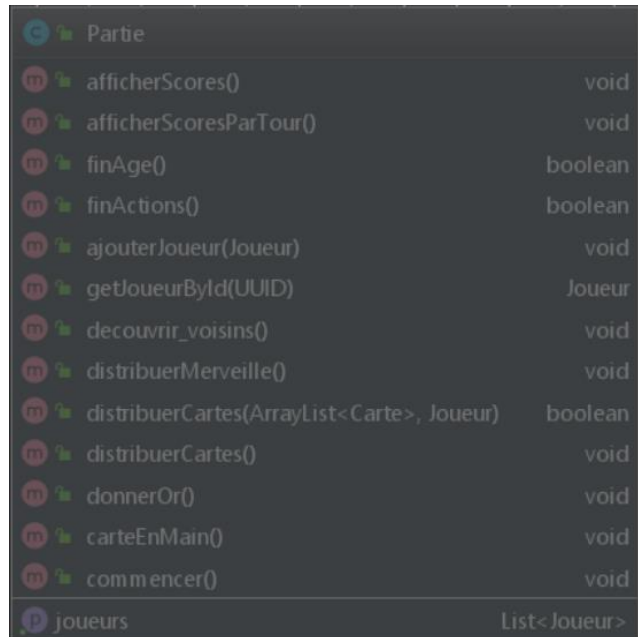
b) Création d'une partie

La classe Partie :

Elle est composée d'une liste de joueurs.

Elle est composée de plusieurs méthodes :

- AfficherScores()
- AfficherScoresParTour()
- finAge()
- finAction()
- ajouterJoueur(UUID) : elle permet d'instancier un nouveau joueur.
- distribuerMerveille() : elle permet d'attribuer de façon aléatoire une merveille à chacun des joueurs.
- distribuerCartes(ArrayList<Carte>, Joueur) : elle permet d'attribuer de façon aléatoire un deck à un joueur choisi, autrement dit, un ensemble de cartes.
- Commencer() : cette méthode va faire appel aux méthodes citées ci-dessus pour permettre le lancement de la partie.
- CarteEnMain() : cette méthode va permettre d'obtenir les cartes en main de chaque joueur



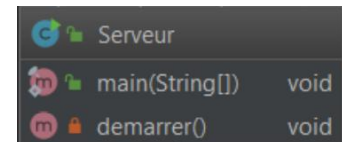
Partie	
afficherScores()	void
afficherScoresParTour()	void
finAge()	boolean
finActions()	boolean
ajouterJoueur(Joueur)	void
getJoueurById(UUID)	Joueur
decouvrir_voisins()	void
distribuerMerveille()	void
distribuerCartes(ArrayList<Carte>, Joueur)	boolean
distribuerCartes()	void
donnerOr()	void
carteEnMain()	void
commencer()	void
joueurs	List<Joueur>

c) Lancement d'une partie

La classe Serveur :

Elle possède plusieurs méthodes, les suivantes :

- Serveur(Configuration) : qui va permettre d'instancier une nouvelle partie, et le serveur SocketIO.
- Main(String[]) : qui va permettre d'afficher les informations utiles quant à l'avancement de la partie.
- Démarrer() : qui va permettre la connexion du serveur, et ainsi de débiter la partie.



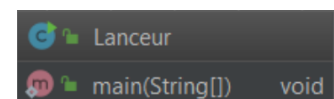
Serveur	
main(String[])	void
démarrer()	void

La classe Lanceur :

Cette classe possède une seule méthode main(String[]).

Cette méthode va servir à créer des threads, autrement

dit les joueurs. Ensuite elle va établir la connexion entre les classes Serveur et Joueur, pour connecter les joueurs à la partie. Dès lors, le serveur sera lancé et paré à lancer le jeu.



Lanceur	
main(String[])	void

d) Les utilisateurs

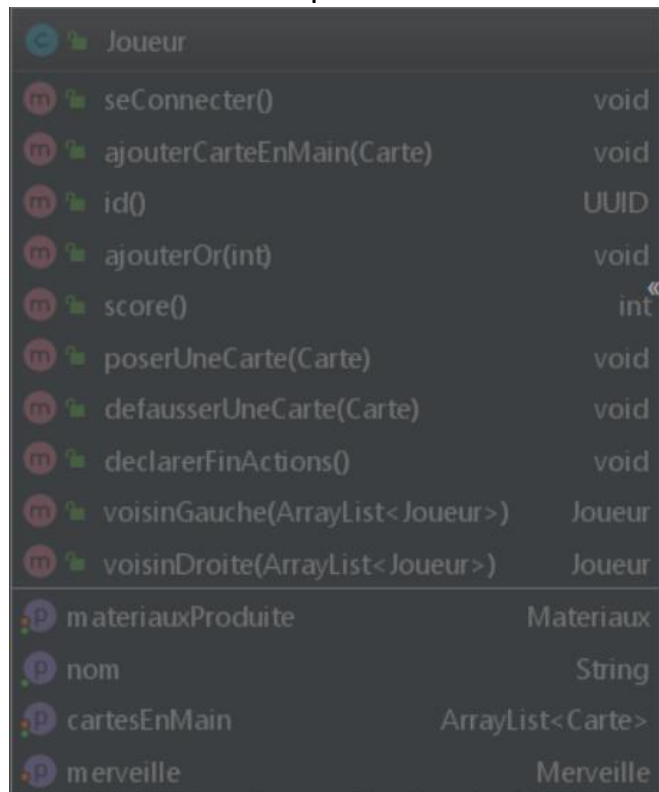
La classe Joueurs est composée de plusieurs caractéristiques dont :

- matériauxProduits
- Nom
- cartesEnMain
- merveille

Puis de méthodes :

- seConnecter()
- ajouterCarteEnMain(Carte)
- id()
- ajouterOr(int)
- score()
- poserUneCarte(Carte)
- defausserUneCarte(Carte)
- declarerFinActions()
- voisinGauche(ArrayList<Joueur>)
- voisinDroite(ArrayList<Joueur>)

L'intérêt de cette classe est de stocker les informations sur les joueurs pendant le déroulement de la partie.

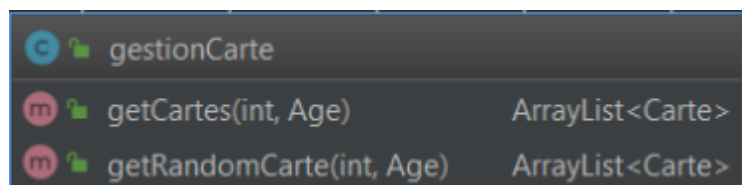


Joueur	
m	seConnecter() void
m	ajouterCarteEnMain(Carte) void
m	id() UUID
m	ajouterOr(int) void
m	score() int
m	poserUneCarte(Carte) void
m	defausserUneCarte(Carte) void
m	declarerFinActions() void
m	voisinGauche(ArrayList<Joueur>) Joueur
m	voisinDroite(ArrayList<Joueur>) Joueur
<hr/>	
p	matériauxProduite Matériaux
p	nom String
p	cartesEnMain ArrayList<Carte>
p	merveille Merveille

e) Gestion des cartes

Cette classe sert à gérer les cartes tout au long du déroulement de la partie grâce aux méthodes :

- getCartes(int, Age)
- getRandomCarte(int, Age)



gestionCarte	
m	getCartes(int, Age) ArrayList<Carte>
m	getRandomCarte(int, Age) ArrayList<Carte>

f) Gestion des merveilles

Cette classe sert à gérer les merveilles et leur distribution avec les méthodes :

- getMerveille(String, Face)
- getRandomMerveille(int)

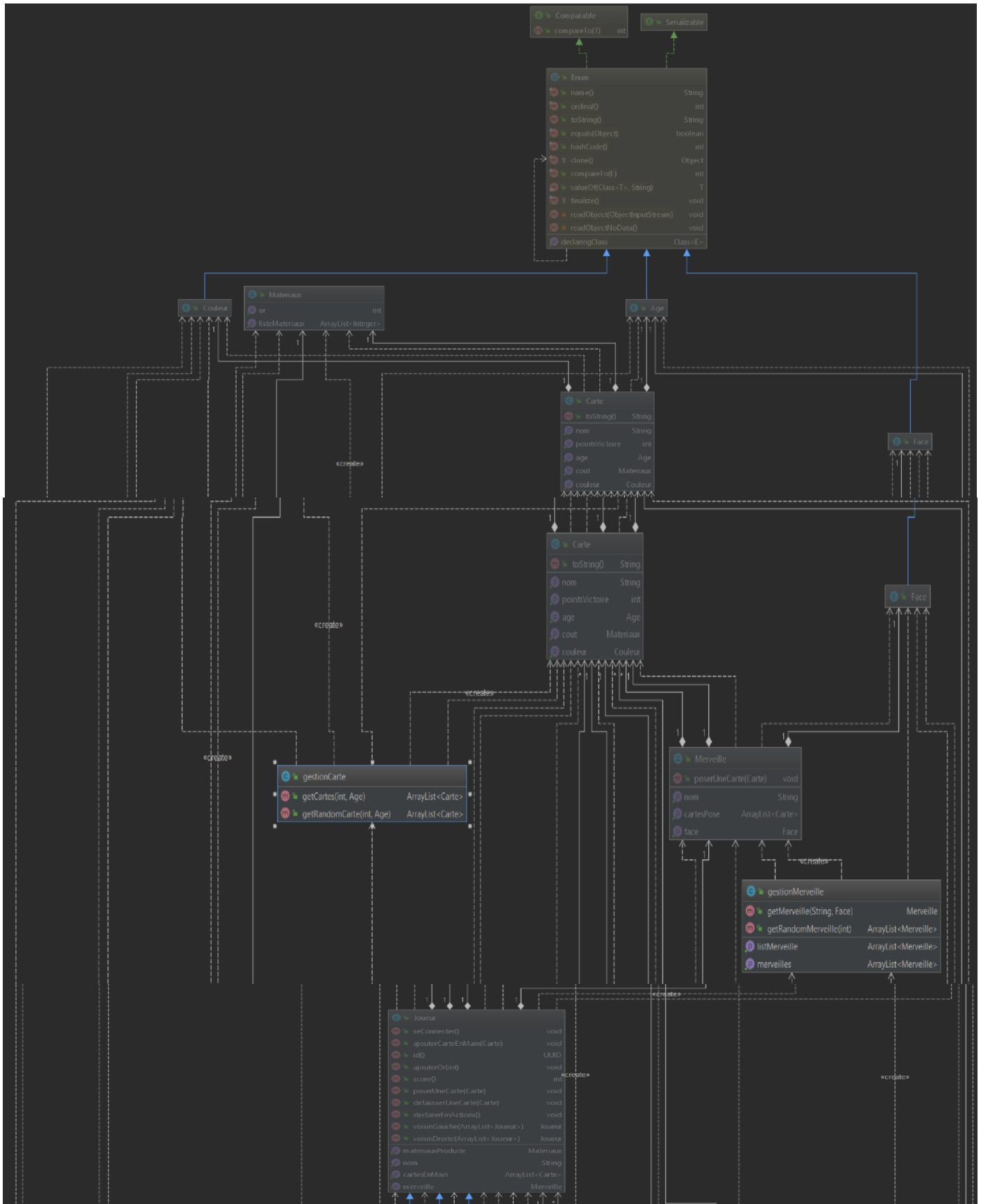
Et les listes suivantes qui vont les stocker :

- listMerveille
- merveilles



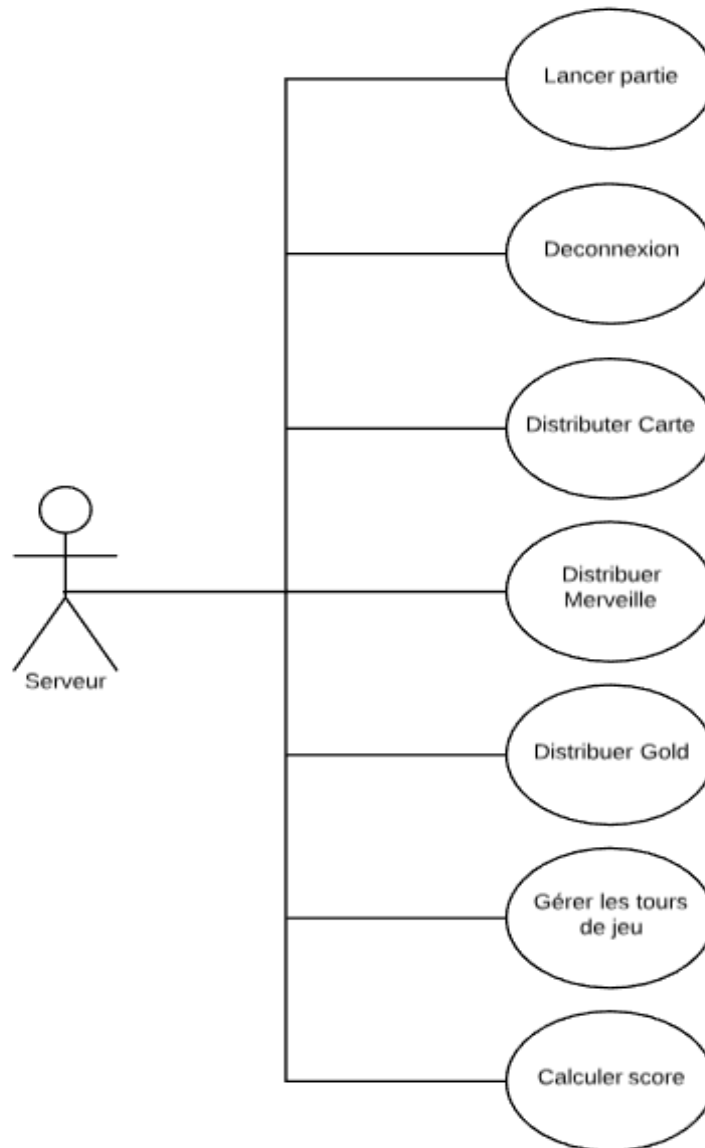
gestionMerveille	
m	getMerveille(String, Face) Merveille
m	getRandomMerveille(int) ArrayList<Merveille>
<hr/>	
p	listMerveille ArrayList<Merveille>
p	merveilles ArrayList<Merveille>

g) Le diagramme en détail



III. USE CASE

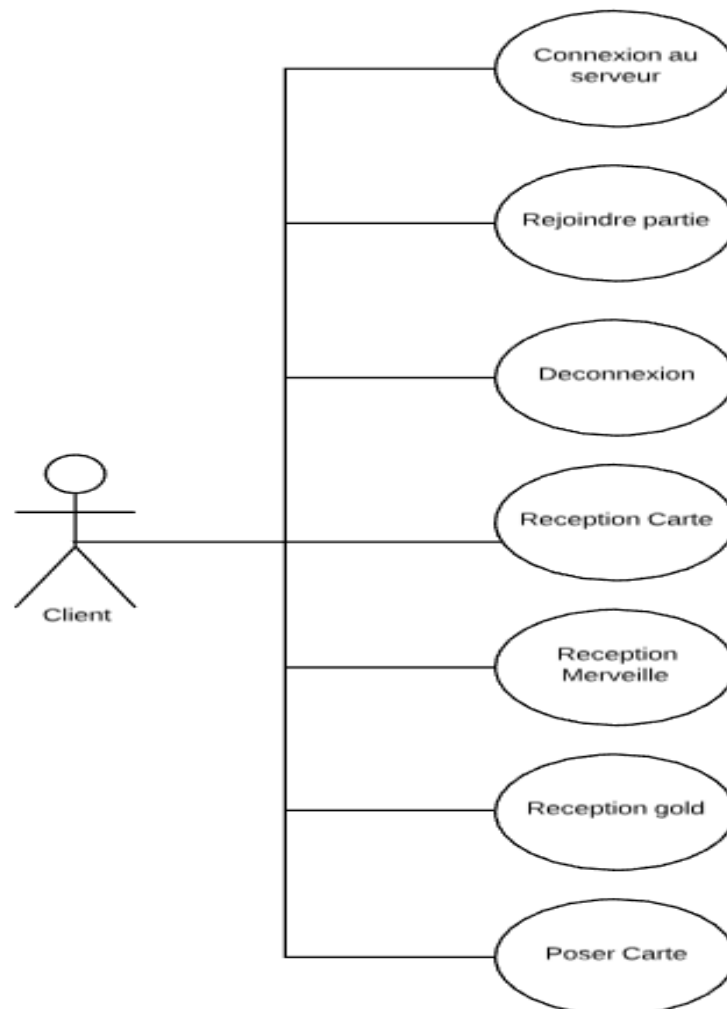
a) Use Case côté client



- Lorsque le serveur est lancé, il va attendre la connexion de joueurs, lorsque c'est fait, celui-ci lancera la partie.
- Les joueurs vont s'identifier puis se verront attribués cartes, merveilles et pièces.

- Durant la partie, le serveur va distribuer les cartes, merveilles, l'or, gérer les tours de jeu lorsque les cartes sont jouées par tous les joueurs et calculer le score à la fin de chaque tour ainsi qu'à la fin de la partie. Lorsque la partie se termine, c'est-à-dire arrivé au calcul de score final, le serveur se déconnecte.

b) Use Case côté serveur

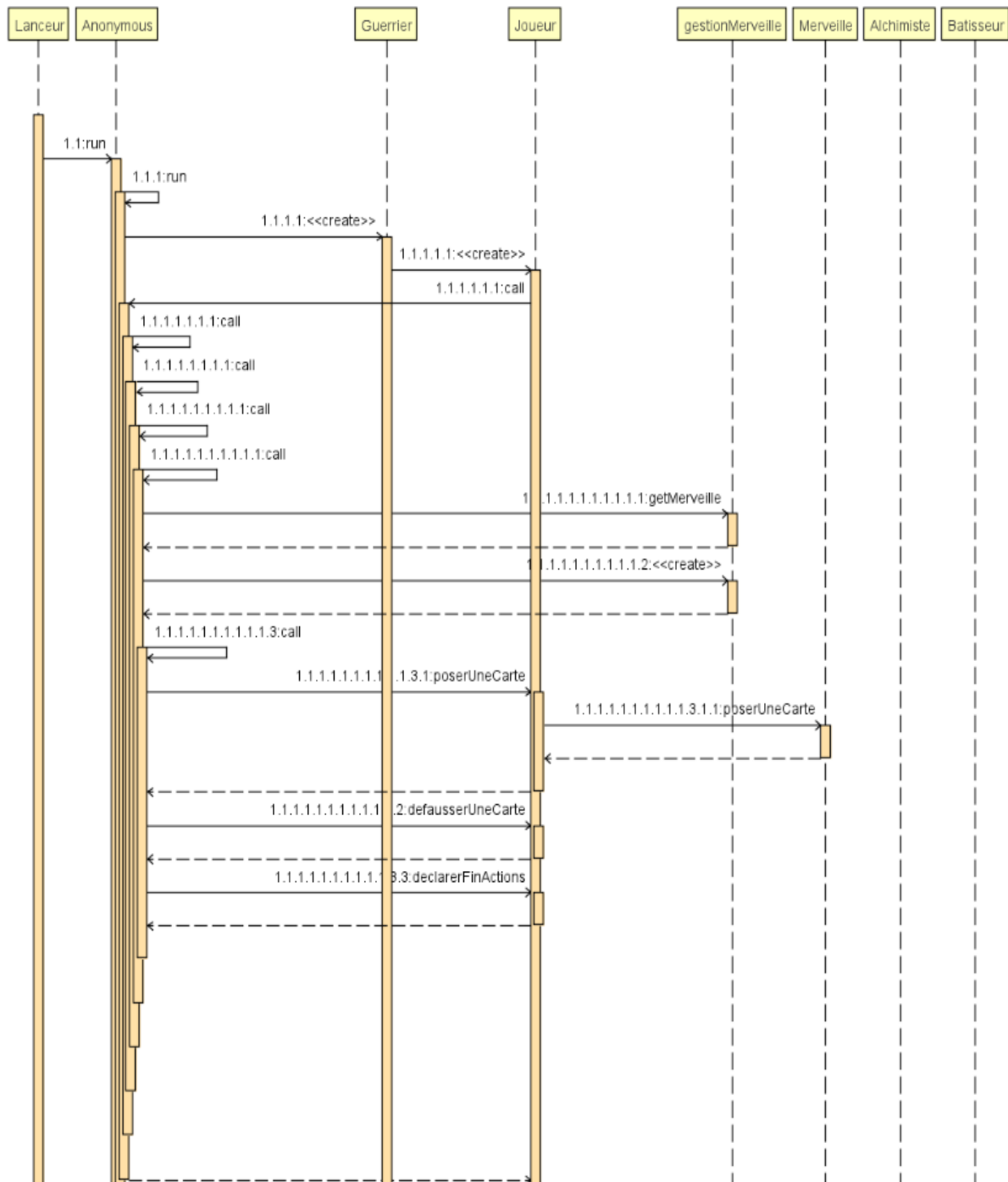


- Les clients vont d'abord se connecter au serveur ensuite émettre un message, en indiquant qu'ils sont bien connectés ainsi que leurs noms.
- Les clients devront attendre l'arrivée de 3 joueurs avant le début de la partie.

- Ils vont recevoir une merveille, 7 cartes et 3 pièces. Les clients vont ensuite attendre un autre signal du serveur leur permettant de commencer à poser leurs cartes.
- Par la suite le serveur enverra des signaux qui permettront aux clients d'effectuer leurs actions, par exemple poser une carte. Une fois la carte posée ils transmettront un signal de fin de tour et le serveur fera faire tourner les cartes des joueurs.
- Toutes les actions des clients sont donc gérées par le serveur qui permet le bon déroulement de la partie.
- Une fois les actions des clients terminées, ceux-ci se déconnectent du serveur. La partie est terminée.

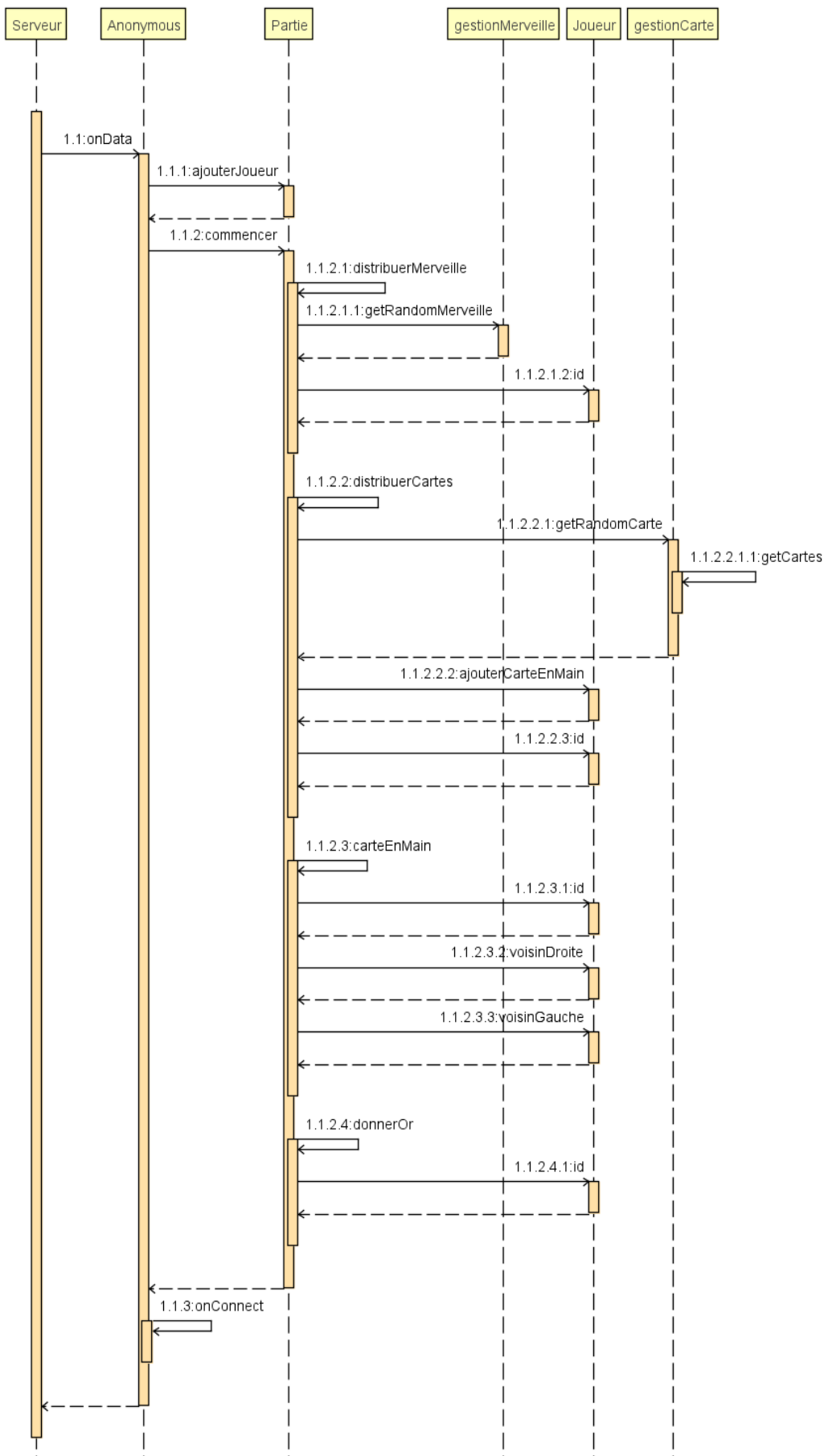
IV. Diagramme séquence

a) Diagramme de séquence du joueur (cf use case cote serveur)



L'acteur ici est le client, qui après avoir reçu les cartes ayant reçu les cartes et les merveilles pourra poser une carte ou défausser. Le client peut avoir trois aspects : guerrier alchimiste ou bâtisseur.

b) Diagramme de séquence du serveur (cf use case côté client)



Le serveur s'exécute et dispose d'un socket lié à un numéro de port. Le serveur est en attente, il attend la connexion d'un client.

Par la suite, le client connaissant le numéro de port ainsi que l'adresse IP du serveur se connecte et s'identifie.

Le serveur accepte la connexion. Ainsi le serveur peut entamer le commencement de la partie.

Ensuite, le serveur ayant la liste des joueurs, créera les merveilles et les distribuera aux joueurs. Puis si tous les joueurs ont reçu une merveille, il entame la distribution des 7 cartes tirées aléatoirement aux joueurs et des pièces d'or.

V. Interaction client serveur

Le serveur s'exécute et dispose d'un socket lié à un numéro de port. Le serveur est en attente, il attend la connexion d'un client.

Par la suite, le client connaissant le numéro de port ainsi que l'adresse IP du serveur se connecte et s'identifie.

Le serveur accepte la connexion. Lors de l'acceptation, le serveur obtient un nouveau socket lié au même port local. Il a besoin d'un nouveau socket pour pouvoir continuer à écouter le socket d'origine pour les demandes de connexion.

Le serveur ajoute une limitation sur le nombre de joueurs.

Du côté du client, si la connexion est accordée, un socket sera créé et le client pourra utiliser ce socket pour avoir la possibilité de communiquer avec le serveur. Le serveur mémorise le joueur, crée la merveille et les cartes et les associe aux joueurs. Le client reçoit ces données sous forme de JSON.

Une fois que le joueur (client) a reçu les cartes, il les joue. Une fois les cartes jouées, le serveur effectue le passage des mains (avec la carte jouée en moins). Chaque joueur reçoit alors les cartes de son voisin.

CONCLUSION

On a montré la conception globale faite pour notre projet, où on a démontré en détails les différents diagrammes, qui permettent de bien comprendre le fonctionnement de chaque élément du projet.

On a mis en avant la mise en œuvre d'un système client-serveur dès le début avec la capacité de son utilisation afin de pouvoir manipuler le jeu.

Nous avons réussi à développer les fonctionnalités principales du jeu telles que distribuer des cartes et merveilles, faire passer les cartes entre les joueurs, calculer les scores....

