

12/04/2019

RAPPORT COO

Présenté par :

NITEKA Lys Ciella, JALAL Réda et NABIL Inès.

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

SOMMAIRE

INTRODUCTION	2
I. Diagramme d'Activité.....	3
II. Diagramme de classe	5
a) Eléments du jeu	7
b) Création d'une partie	8
c) Lancement d'une partie	8
d) Les utilisateurs	9
e) Gestion des cartes.....	9
f) Gestion des merveilles	9
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	16
b) Diagramme de séquence du serveur.....	16
CONCLUSION	18

INTRODUCTION

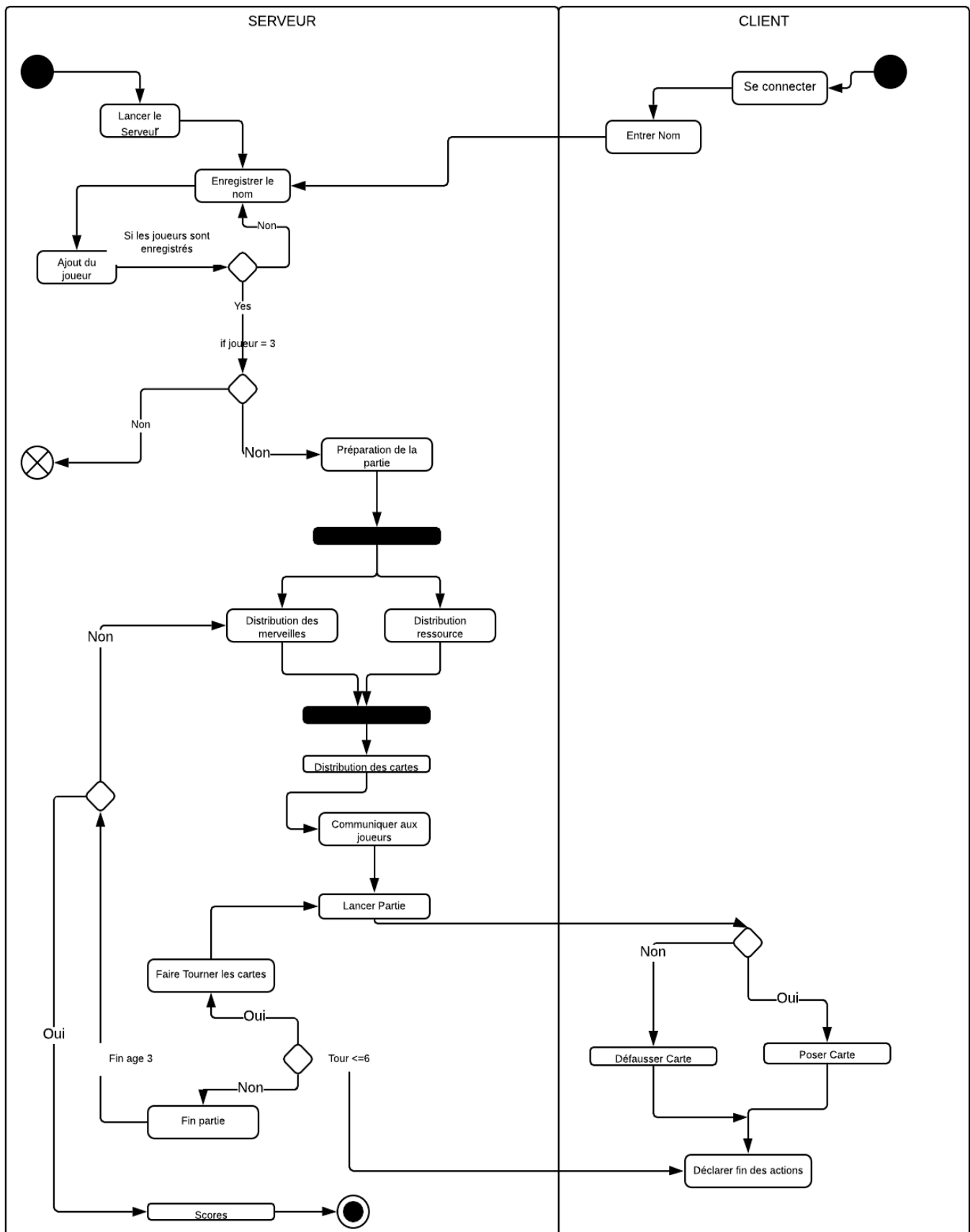
"Seven Wonders" est un jeu dont le thème est les sept merveilles du monde. C'est un jeu évolutif ayant pour but la construction des bâtiments des sept merveilles. Le jeu se joue jusqu'à sept joueurs. La main de départ contient sept cartes. Les joueurs développent leur civilisation autour de sept types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

→ 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.

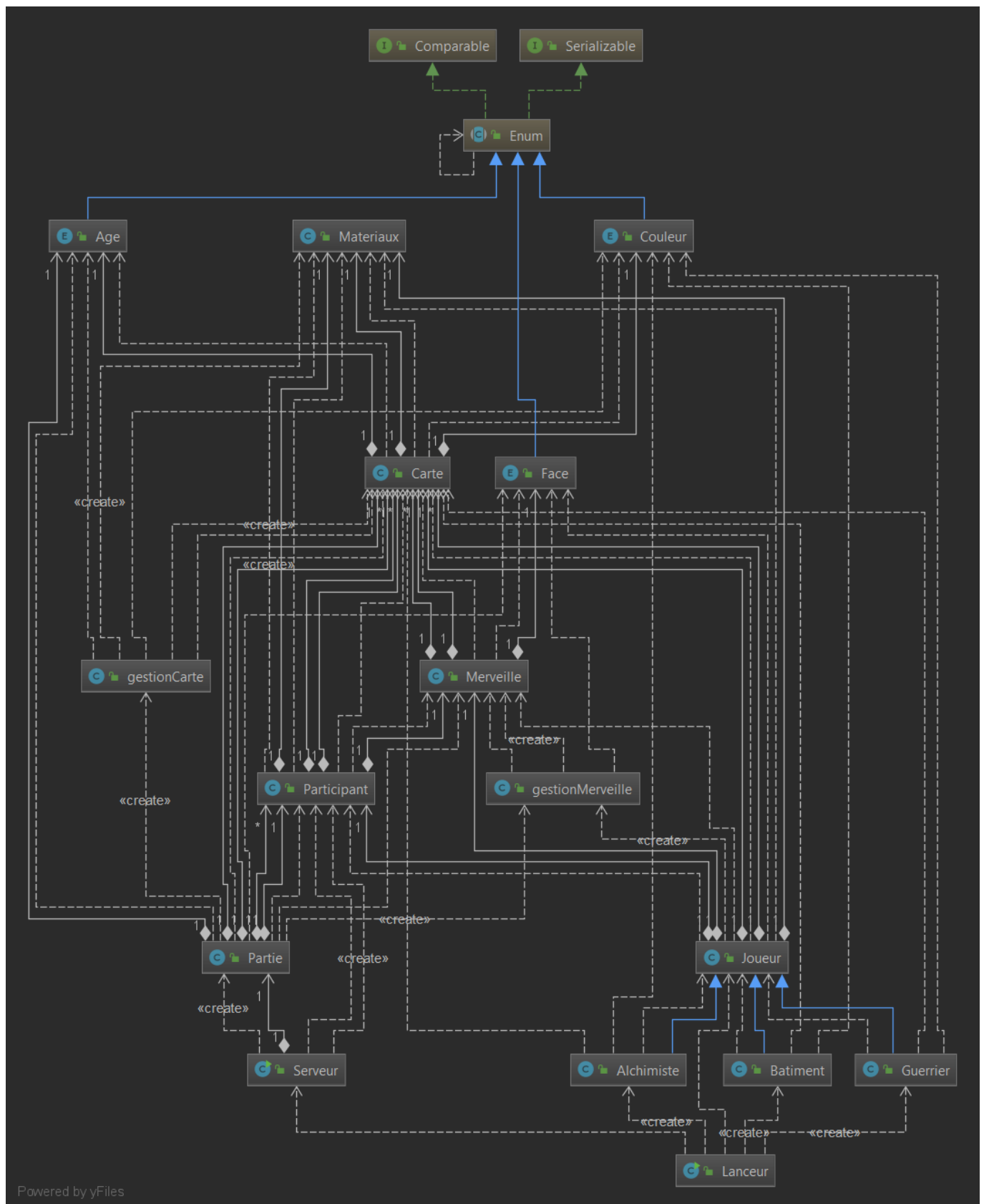
→ A la fin de la partie, il affiche les scores et termine ses actions.

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.

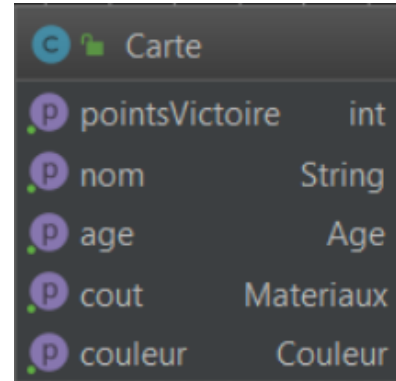


a) Éléments du jeu

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)

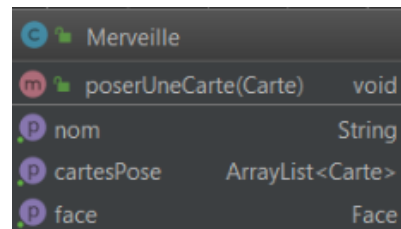


C	Carte	
P	pointsVictoire	int
P	nom	String
P	age	Age
P	cout	Matériaux
P	couleur	Couleur

La classe Merveille :

Elle est constituée de plusieurs caractéristiques, les suivantes :

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

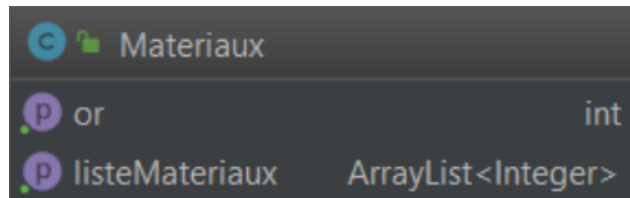


C	Merveille	
M	poserUneCarte(Carte)	void
P	nom	String
P	cartesPose	ArrayList<Carte>
P	face	Face

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 :

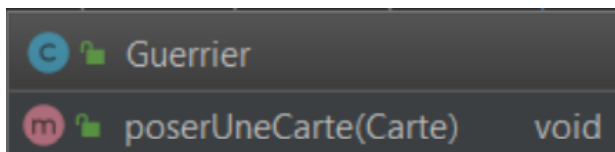
Cette classe contient les différents matériaux contenus dans le jeu, qui seront stockés dans une liste.



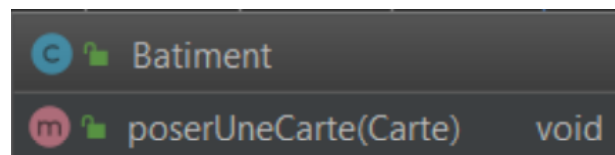
C	Matériaux	
P	or	int
P	listeMateriaux	ArrayList<Integer>

Les classes Guerrier, Batiment, Alchimiste :

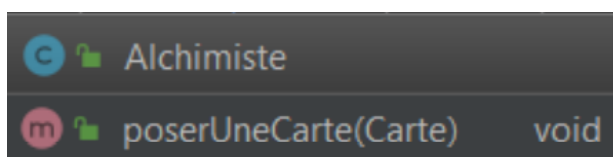
Ce sont les différents types de profils des joueurs auxquels on aura assigné les cartes qui leurs correspondent.



C	Guerrier	
M	poserUneCarte(Carte)	void



C	Batiment	
M	poserUneCarte(Carte)	void



C	Alchimiste	
M	poserUneCarte(Carte)	void

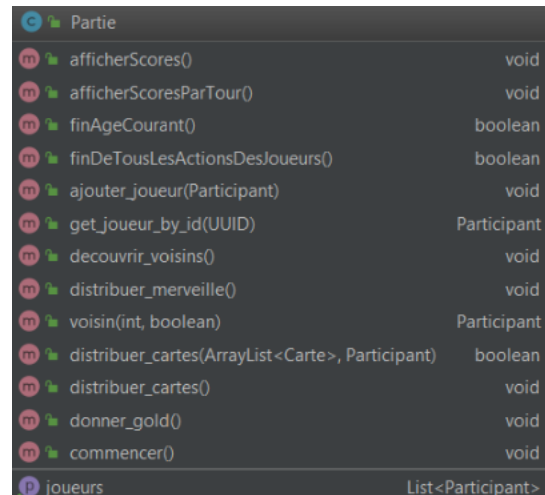
b) Création d'une partie

La classe Partie :

Elle est composée d'une liste de joueurs.

Elle est composée de plusieurs méthodes :

- AfficherScores()
- AfficherScoresParTour()
- finAgeCourant()
- finDeTousLesActionsDesJoueurs()
- Ajouter_joueur(UUID) : elle permet d'instancier un nouveau joueur.
- Distribuer_merveille() : elle permet d'attribuer de façon aléatoire une merveille à chacun des joueurs.
- Distribuer_cartes(ArrayListe<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.



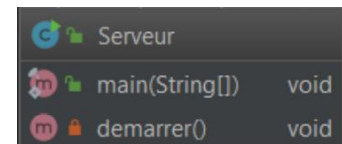
Partie	
afficherScores()	void
afficherScoresParTour()	void
finAgeCourant()	boolean
finDeTousLesActionsDesJoueurs()	boolean
ajouter_joueur(Participant)	void
get_joueur_by_id(UUID)	Participant
decouvrir_voisins()	void
distribuer_merveille()	void
voisin(int, boolean)	Participant
distribuer_cartes(ArrayList<Carte>, Participant)	boolean
distribuer_cartes()	void
donner_gold()	void
commencer()	void
joueurs	List<Participant>

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.



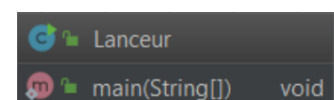
Serveur	
main(String[])	void
demarrer()	void

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

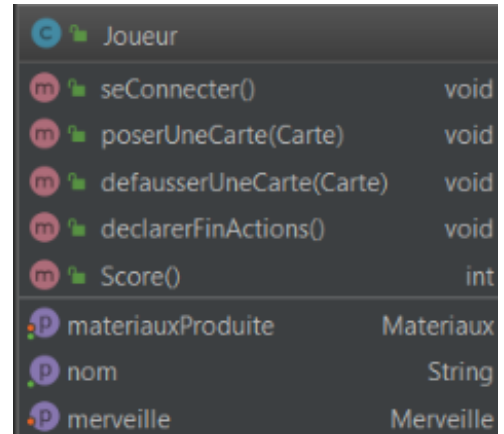
d) Les utilisateurs

La classe Joueurs est composée de plusieurs caractéristiques dont :

- materiauxProduite
- nom
- merveille

Puis de méthodes :

- seConnecter()
- poserUneCarte(Carte)
- defausserUneCarte(Carte)
- declarerFinActions()
- Score()



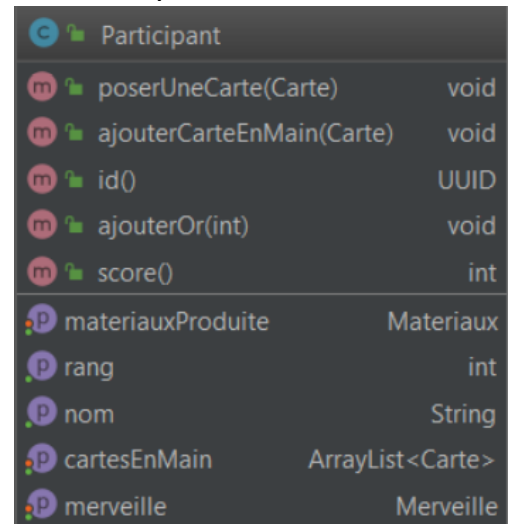
Joueur		
m	seConnecter()	void
m	poserUneCarte(Carte)	void
m	defausserUneCarte(Carte)	void
m	declarerFinActions()	void
m	Score()	int
p	materiauxProduite	Materiaux
p	nom	String
p	merveille	Merveille

La classe Participant est composée de plusieurs caractéristiques dont :

- materiauxProduite
- rang
- nom
- cartesEnMain
- merveille

Et de plusieurs méthodes :

- poserUneCarte(Carte)
- ajouterCarteEnMain(Carte)
- id()
- ajouterOr(int)
- score()



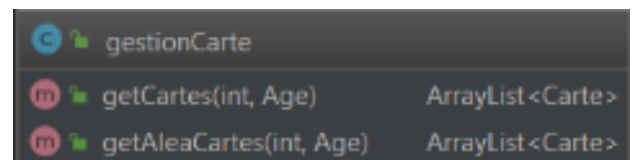
Participant		
m	poserUneCarte(Carte)	void
m	ajouterCarteEnMain(Carte)	void
m	id()	UUID
m	ajouterOr(int)	void
m	score()	int
p	materiauxProduite	Materiaux
p	rang	int
p	nom	String
p	cartesEnMain	ArrayList<Carte>
p	merveille	Merveille

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

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)
- getAleaCartes(int, Age)



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

f) Gestion des merveilles

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

- getMerveille(String, Face)

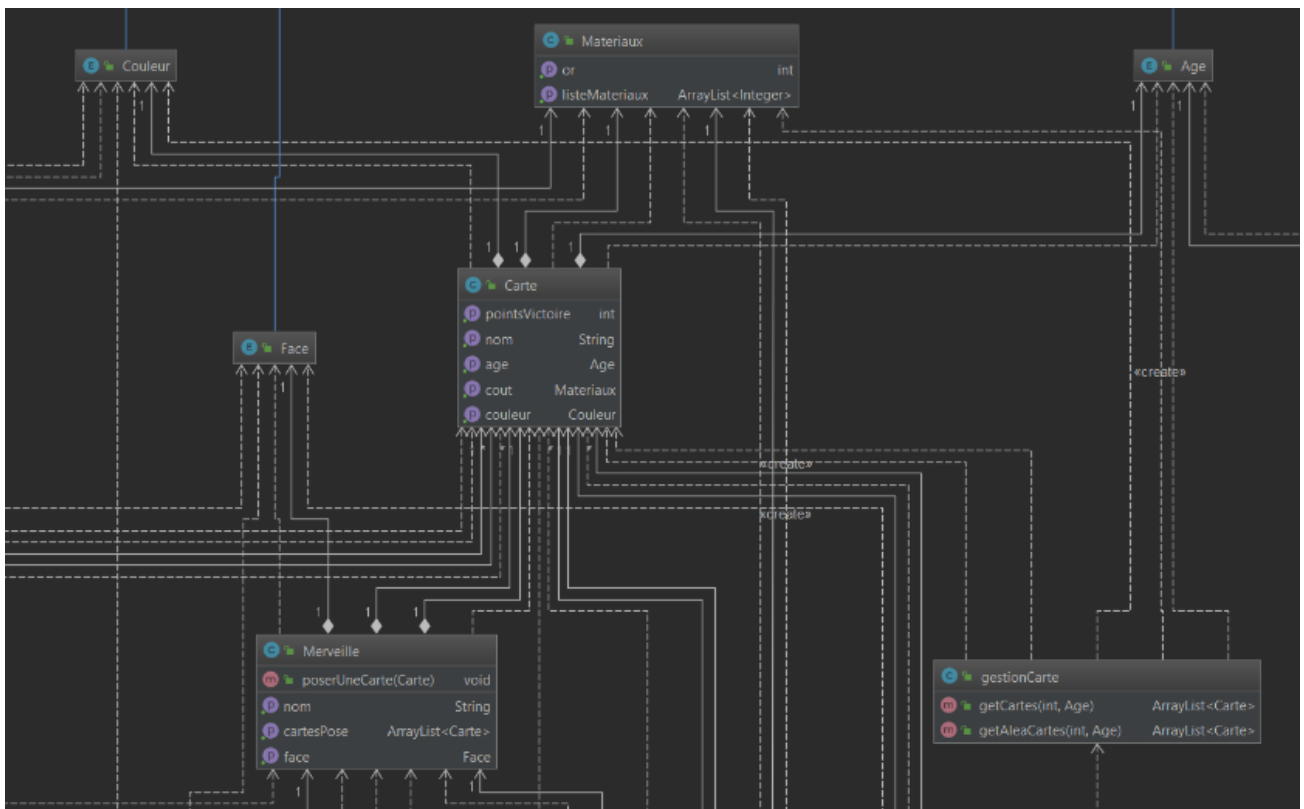
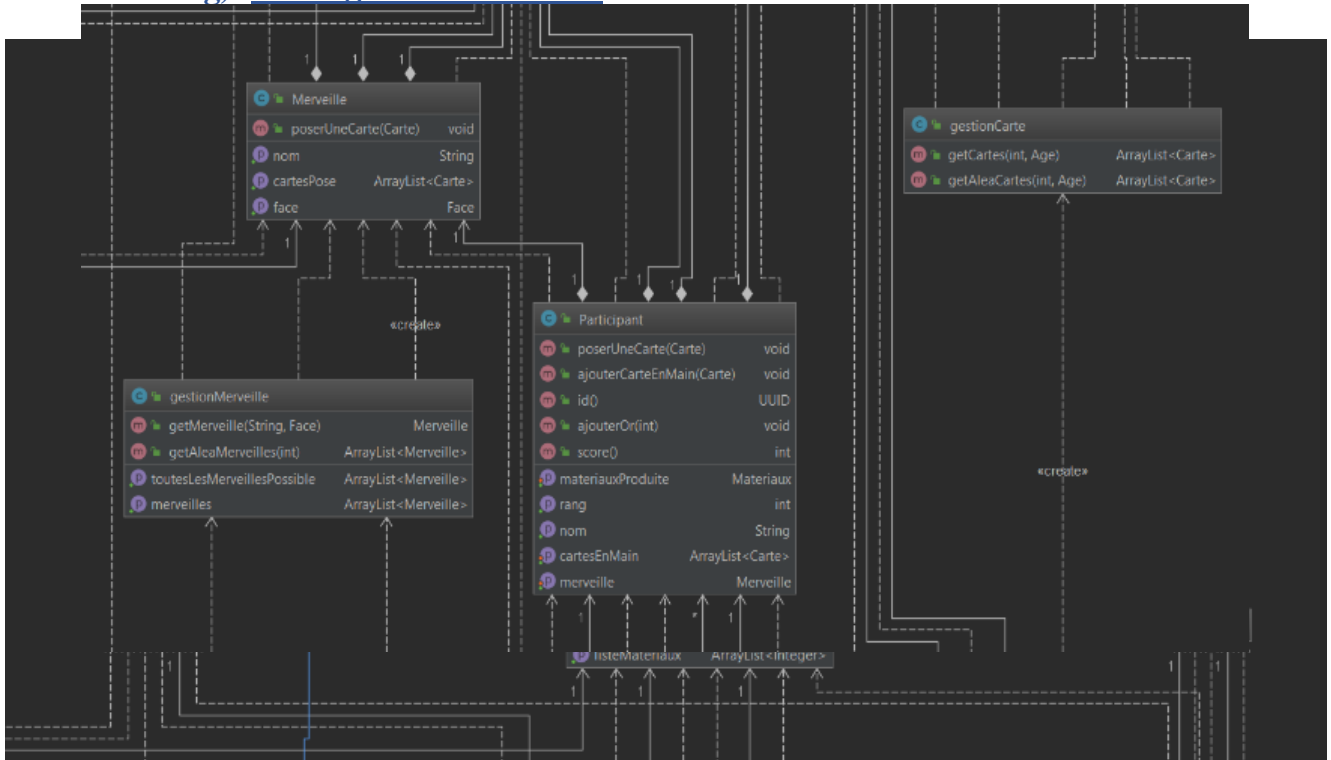
- getAleaMerveilles(int)

Et les listes suivantes qui vont les stocker :

- toutesLesMerveillesPossible
- merveilles

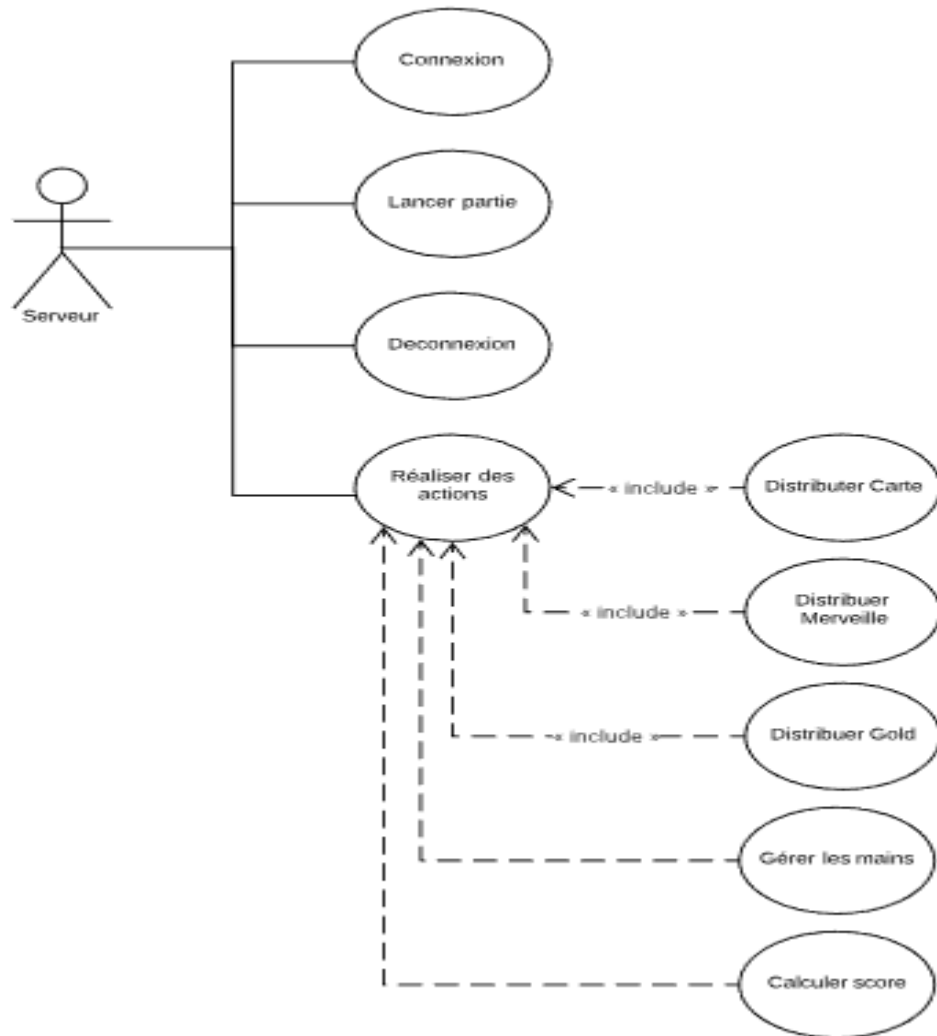
 gestionMerveille	
 getMerveille(String, Face)	Merveille
 getAleaMerveilles(int)	ArrayList<Merveille>
 toutesLesMerveillesPossible	ArrayList<Merveille>
 merveilles	ArrayList<Merveille>

g) Le diagramme en détail



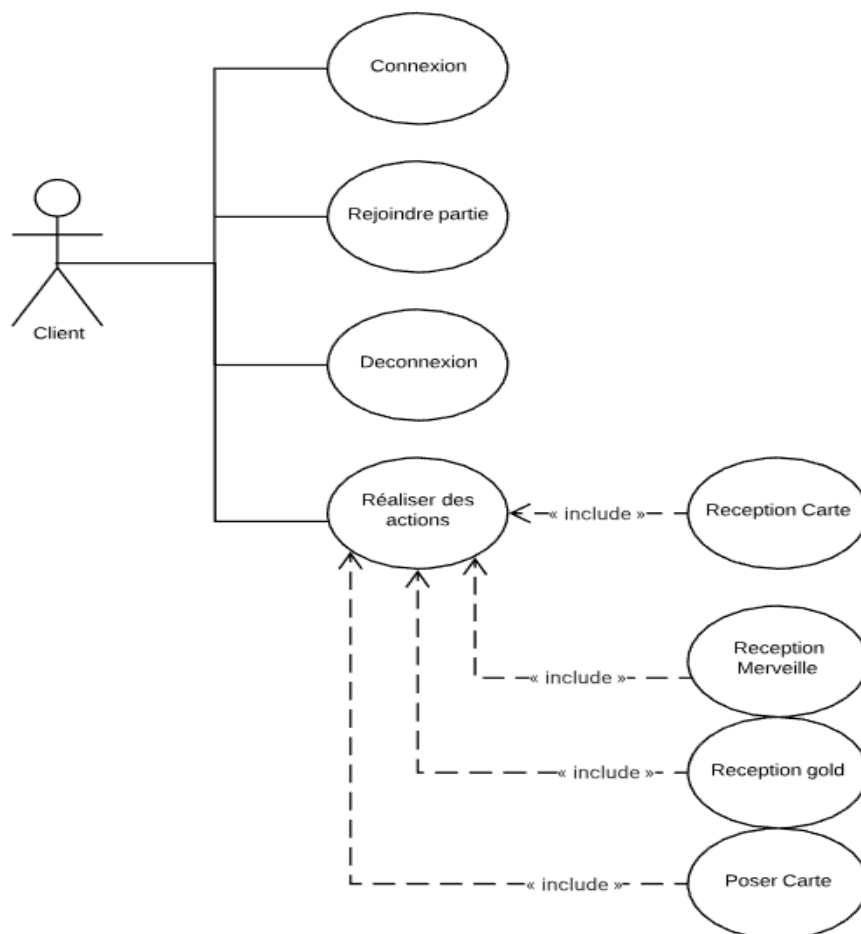
III. USE CASE

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 réalisera des actions qui permettent l'organisation d'une partie telles que distribuer des cartes, gérer les mains lorsque les cartes sont jouées, calculer le score....
- Lorsque toutes les actions ont été réalisées, les joueurs se déconnecteront puis le serveur se déconnectera.

b) Use Case côté serveur

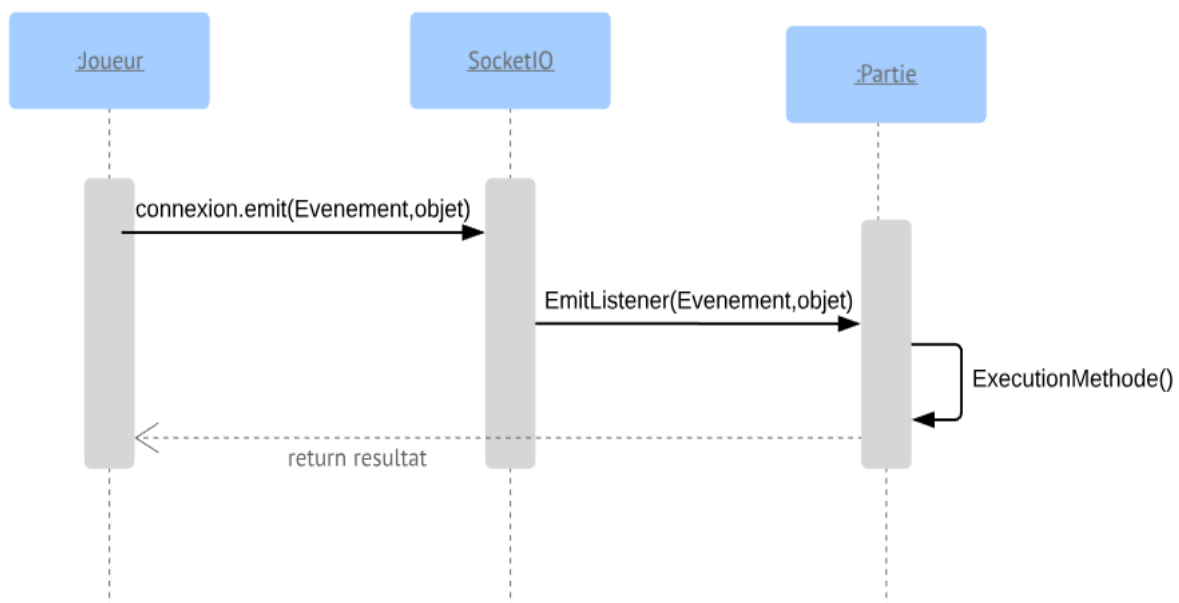


- Les clients vont d'abord se connecter au serveur ensuite émettre un message, en indiquant qu'ils sont bien connectés ainsi que leurs noms.
- Les clients devront attendre l'arrivée de 3 joueurs avant le début de la partie.
- Ils vont recevoir une merveille, 7 cartes et 3 pièces. Les clients vont ensuite 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

- Le joueur envoie un message grâce à la fonction emit située dans la librairie socketIO.
- Le message aura pour paramètre le nom de l'évènement ainsi que l'objet correspondant à envoyer.
- Le serveur par le biais d'un Listener recevra le signal de l'évènement puis exécute des méthodes et renvoie le résultat.

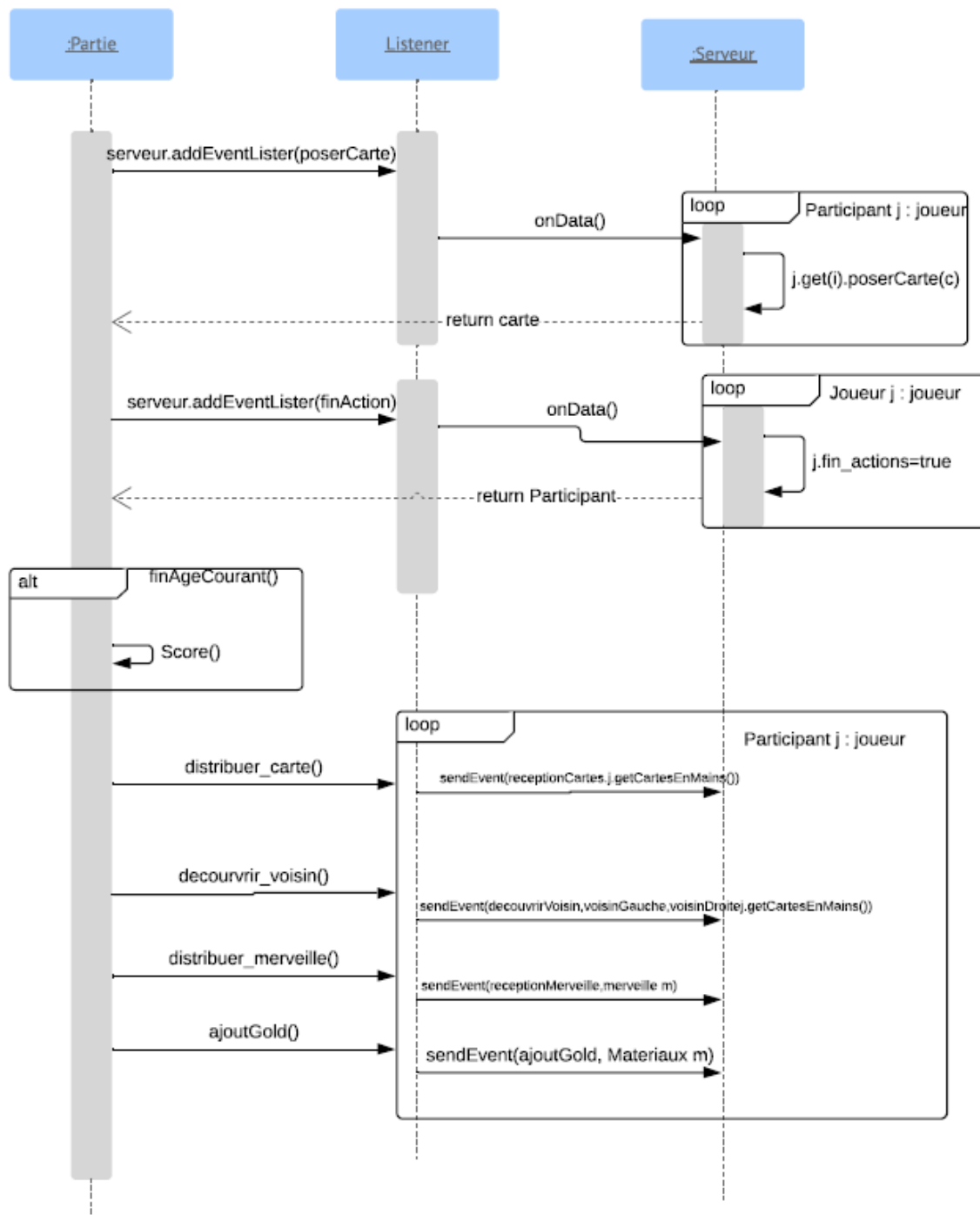


b) Diagramme de séquence du serveur

Le diagramme proposé décrit les étapes lors de la communication entre le serveur et la partie les étapes sont les suivantes :

- Lorsque la partie est créée, on ajoute un évènement poser une carte, le serveur retourne alors la carte posée par les tous joueurs avec le biais d'une boucle. Puis un évènement `finAction` qui retourne le joueur ayant fini ses actions

- Si nous sommes à la fin des âges, la méthode score est exécutée.
- Viens ensuite la distribution des cartes, merveilles, gold qui seront réceptionnées par tous les joueurs ainsi que la découverte des voisins qui servira pour le passage des cartes lors de la fin des tours.



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.

On consacrera plus de temps à mieux améliorer les différents tests unitaires.