UNIVERSITÉ CÔTE D'AZUR

miage NICE SOPHIA ANTIPOLIS

Les bâtisseurs du moyen-âge

8 Janvier 2021
Découverte Métier

Mathieu RETHERS, Yannick ALCARAZ, Nicolas CORBIERE, Alexandre LEMOINE & Lisa DESLANDES

Mr RENEVIER GONIN

Table des matières

Introduction

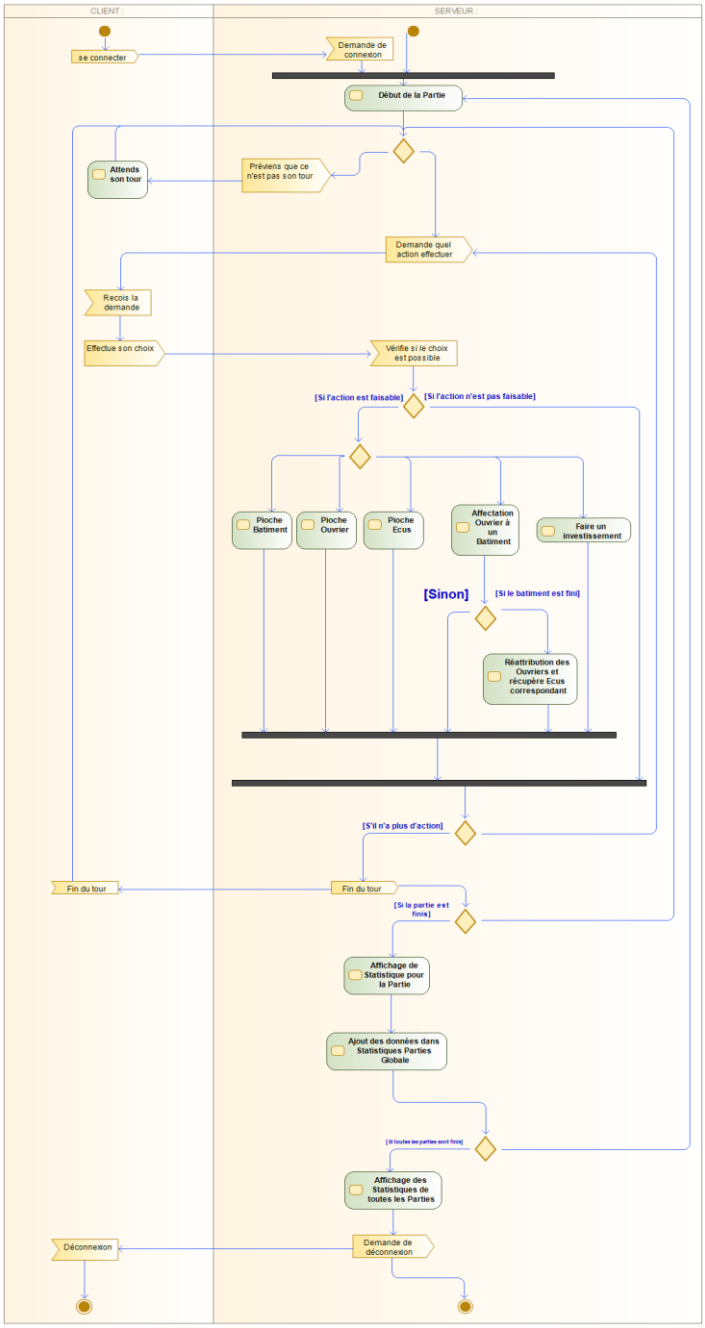
Ce rapport provient d'un projet réalisé dans le cadre de notre cursus Licence 3 Parcours MIAGE. Le sujet de ce projet est de réaliser une version informatique du jeu de cartes et de stratégie : Les bâtisseurs du Moyen-Âge. Le but du jeu est simple : nous avons des cartes ouvriers et bâtiments. Notre but est de finir un maximum de carte bâtiment jusqu'à 17 points de victoires.

Ce projet, fait en groupe, nous permet ainsi de développer nos compétences en équipes, mais aussi en professionnel. Nous épanouissant ainsi dans beaucoup de domaines tel que la gestion du git, de l'organisation d'un groupe et de la mise en place de normes interne telle que celle de nommages par exemple.

Point de vue général

Diagramme d'activité

Application dans le sujet



Méthodes fonctionnelles

Moyen-Âge

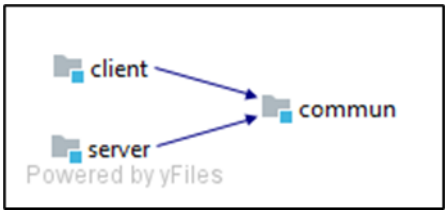
Label	Fait ? (o/n)
Toutes les cartes sont dans la partie	Oui
Les différentes piles de cartes sont mélangées	Oui
Les joueurs reçoivent une carte "apprenti" au début de la partie	Oui
Les joueurs peuvent voir les 5 premières cartes bâtiments	Oui
Les joueurs peuvent voir les 5 premières cartes ouvriers	Oui
Les joueurs reçoivent 10 écus en début de partie	Oui
À chaque début de tour, les joueurs ont 3 actions	Oui
Le joueur peut ouvrir un chantier	Oui
Le joueur peut recruter un ouvrier	Oui
Le joueur envoyer travailler un ouvrier	Oui
Le joueur peut envoyer travailler plusieurs ouvriers contre le bon nombre d'actions	Oui
Le joueur peut prendre un ou plusieurs écus	Oui
Le joueur peut acheter une action	Oui
À la fin d'un chantier, le joueur peut récupérer ses ouvriers ainsi que les écus gagnés	Oui
Les machines fonctionnent	Oui
La partie se termine dès qu'un joueur à 17 point de victoire sans les écus	Oui
On ajoute les points de victoire obtenue par les écus	Oui

Antiquité

Label	Fait ? (o/n)
Toutes les cartes sont dans la partie	Oui
Les différentes piles de cartes sont mélangées	Oui
Les joueurs reçoivent une carte "apprenti" au début de la partie	Oui
Les joueurs peuvent voir les 5 premières cartes bâtiments	Oui
Les joueurs peuvent voir les 5 premières cartes ouvriers	Oui
Les joueurs peuvent voir la première carte des Esclaves, Outils, Emprunts et Universités	Oui
Les joueurs reçoivent 10 sesterces en début de partie	Oui
À chaque début de tour, les joueurs ont 3 actions	Oui
Le joueur peut ouvrir un chantier	Oui
Le joueur peut recruter un ouvrier	Oui
Le joueur peut réaliser un investissement	Non, car pas fonctionnel
Le joueur envoyer travailler un ouvrier	Oui
Le joueur peut envoyer travailler plusieurs ouvriers contre le bon nombre d'actions	Oui
Le joueur peut prendre un ou plusieurs sesterces	Oui
Le joueur peut acheter une action	Oui
À la fin d'un chantier, le joueur peut récupérer ses ouvriers ainsi que les sesterces gagnés	Oui
Les machines fonctionnent	Oui
La partie se termine dès qu'un joueur à 17 points de victoire sans les sesterces	Oui
On ajoute les points de victoire obtenue par les sesterces	Oui

Introduction

Afin de créer un service en ligne, soit client/server, nous avons créé 3 modules, alors que l'ancienne version était seulement un seul module contenant tout les classes.



Le module client qui contient seulement les fonctionnements cotés client (connexion vers server, stratégie, etc…), le module server qui contient seulement les fonctionnalités du server (Partie, MoteurTMP, etc…) et le module commun qui possède des fonctionnalités communes client/server.

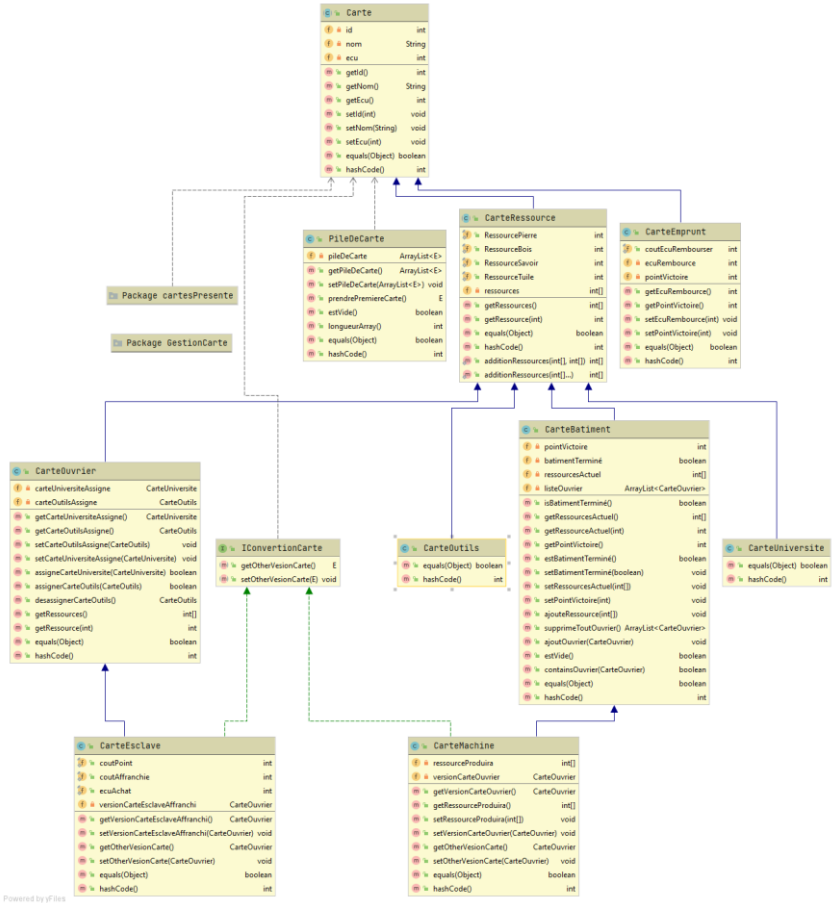
Le module server dépend du module commun et le module client dépend du module commun.
De plus, des fonctionnalités existantes ont été modifiées ou supprimées et de nouvelles fonctionnalités ont été ajoutées.

Commun

Le module commun permet de centraliser les fonctionnalités communes du client et du server afin d'éviter des duplications de code inutile.

Package carte

Le package carte possède des class qui concerne les Cartes afin de représenter des cartes jouables au cours d'une partie. Après l'évolution du projet, des cartes ont été rajoutées comme *CarteEsclave, CarteUniversité, CarteEmprunt, CarteOutils*. De plus, la classe *CarteRessource* a été rajoutée afin de regrouper les cartes ayant des fonctionnalités aux tours des ressources. Nous pouvons noter que le package *carte.cartesPresente* a été ajouté.



Package carte.cartesPresente

Afin de représenter des cartes visibles et proposées dans le jeu, une classe a été créée pour faciliter cette demande, « *CartesPresente* », cette classe générique, qui stockent par défaut des objets de la classe *Carte*, permet de stocker des cartes qui seront lisibles et permet de choisir des cartes.

De plus, les classes *LesCartesPresentes* permettent de ranger des *CartesPresente* en fonction de l'instanciation choisie.

(Voir annexes, « Diagrammes module Commun, Package carte.cartesPresente »)

Package mainJoueur

Afin de représenter les mains des joueurs, un package *mainJoueur* a été ajouté avec une interface *IMainCarte* et une classe *MainJoueur*. La classe générique *MainJoueur*, permet de représenter une main d'un joueur. De plus elle peut être typée selon la classe de la carte.

(Voir annexes, « Diagrammes module Commun, Package mainJoueur »)

Package operation

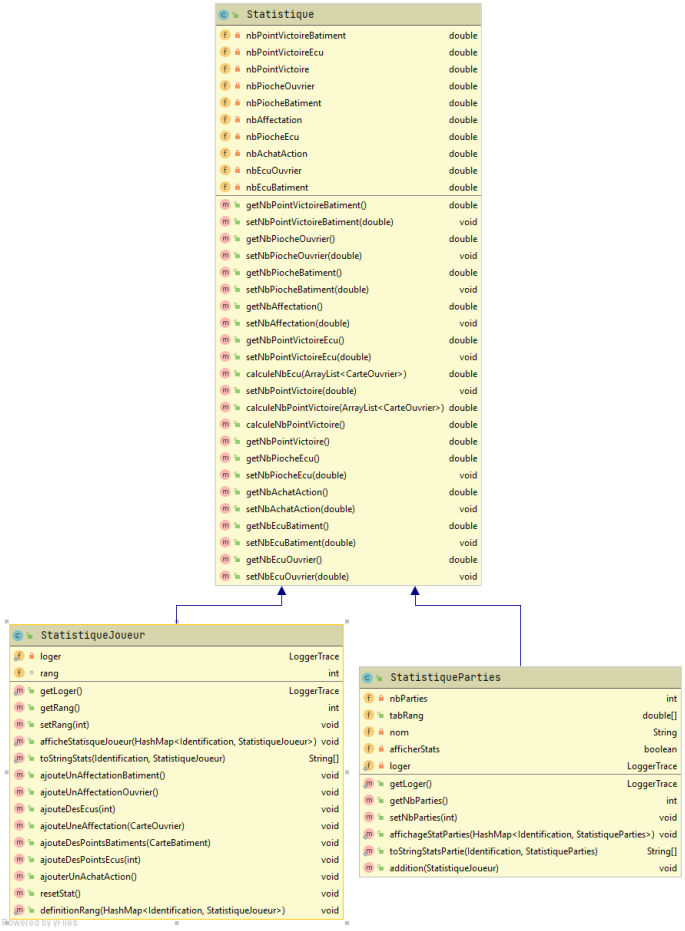
Ce package existant dans l'ancienne version, a été modifier afin de s'adapter à évolution demandée dont l'ajout d'action du joueur possible et spécifique soit les Operations Investissement (*OpEmprunt*, *OpAchatOutil*, *OpRembourserEmprunt*, *OpInstruire*, *OpAffranchirEsclave*, *OpAchatEsclave*). Les opérations existantes ont été légèrement modifiées.

(Voir annexes, « Diagrammes module Commun, Package operation »)

Package statistique

Ce package existant dans l'ancienne version possédée une seule classe *Statistique*, maintenant elle voit apparaître 2 classes filles de la classe *Statistique*, *StatistiqueJoueur* et *StatistiquePartie*.

StatistiqueJoueur permet d'enregistrer les données de la partie effectuée par un joueur. *StatistiquePartie* permet d'enregistrer l'ensemble des données d'une partie.



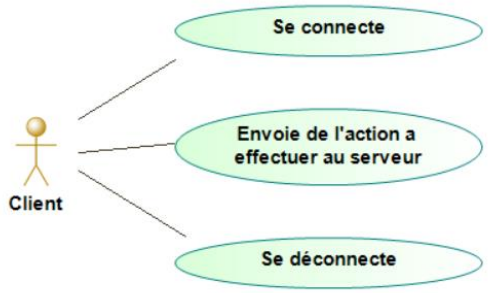
Autre changement

- Ajout de l'interface *MoteurDeJeu*.
- Ajout de l'enum *ModeDeJeu* qui permet de différencier les modes de jeu (Moyen-Age et Antiquité).
- Ajout de la classe statique *Message*.
- Ajout du package *nommage* permet d'afficher sur les consoles des nommages différents selon le mode de jeu choisi.
- Ajout de la classe *InventaireJoueur* qui permet de stocker tous les objets que le joueur peut avoir et aura.
- Ajout de la classe *Identification* permet d'identifier le joueur coté client et server.

Client / Joueur

Analyse des besoins

Diagramme de use case



Scénarios

Nom : Jouer
Acteur(s) : Client
Description : La réalisation d'une action doit être possible pour un client
Auteur : Yannick ALCARAZ
Date(s) : 30/12/2020

Pré-conditions :
4 clients doivent être connectés afin de pouvoir lancer une partie

Démarrage :
L'utilisateur a demandé de lancer le jeu

DESCRIPTION :
Le scénario nominal :

1. L'utilisateur ouvre un terminal au sein de l'IDE
2. L'utilisateur créer un client
3. Le client se connecte à la partie
4. Le client attend un signal du serveur pour jouer
5. Le client réceptionne un signal provenant du serveur lui indiquant qu'il peut jouer
6. Le client choisit une action à effectuer
7. Le client indique au serveur l'action qu'il souhaite effectuer
8. Le client gagne la partie
9. Le client se déconnecte

Les scénarios alternatifs :
8a. Le client n'arrive pas à finir une partie, car l'ensemble des cartes ont été utilisées et le nombre de points de victoire à atteindre pour gagner n'a pas été atteint.
Post-conditions : Aucun

Conception logicielle

Diagrammes de Classes

Le module client permet de séparer les fonctionnalités exclusives du client afin que les modules communs et server ne puissent pas y accéder.

(Voir annexes, « Diagrammes modules, Dernière version » et « Diagrammes module Client, Générale »)

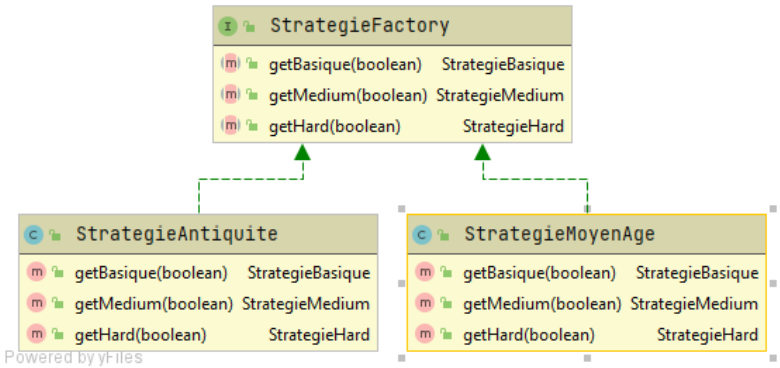
Package stratégie

Package existant dans l'ancienne version, qui s'est vu s'ajouter des classes par stratégie, selon le mode de jeu choisi.

(Voir annexes, « Diagrammes module Client, Package stratégie »)

Package stratégieFactory

Package contenant des classes permettant de générer différents types de stratégie en fonction du mode de jeu choisi.

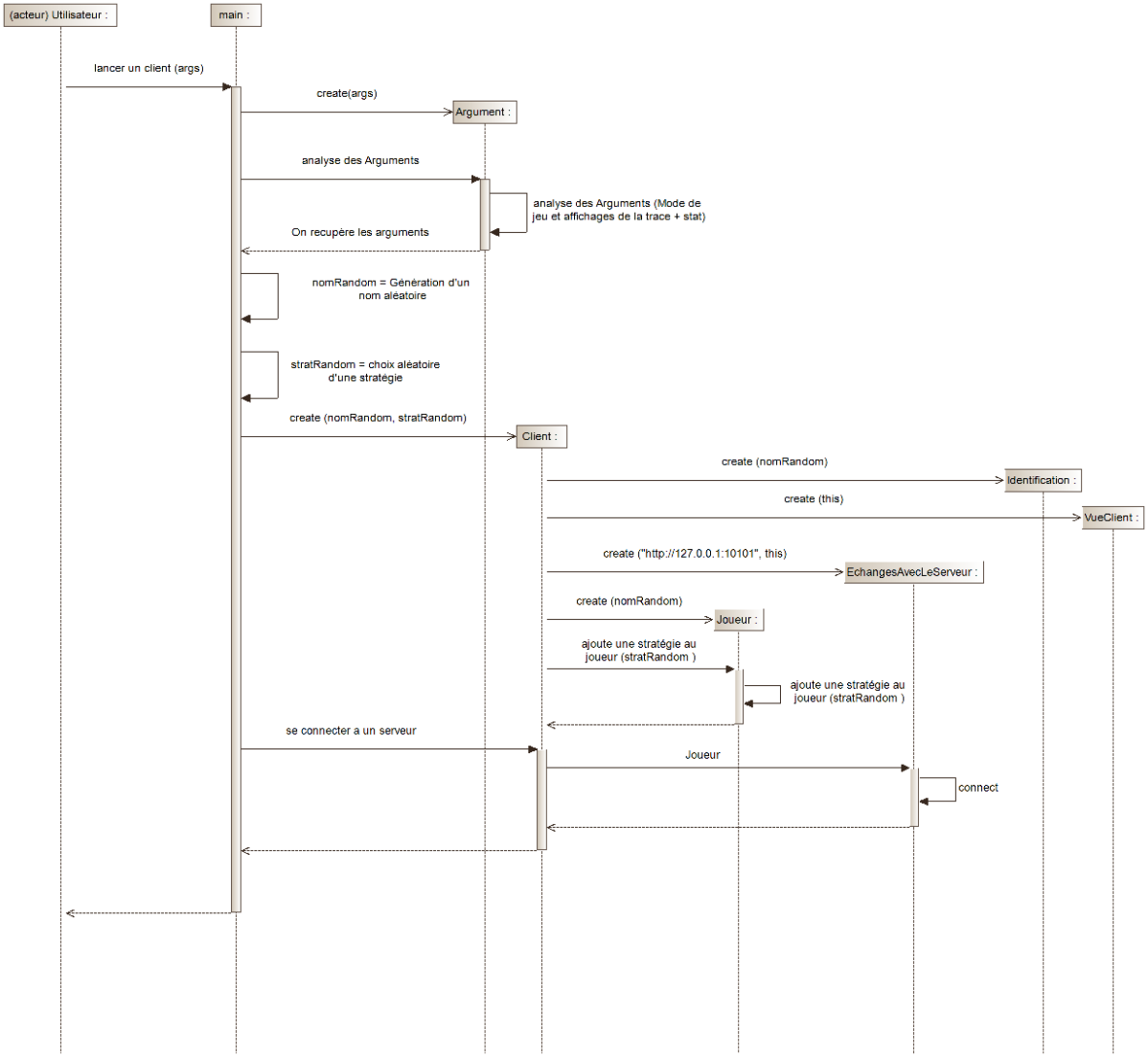


Autre changement

- Ajout de la classe *Client* qui permet de paramétrer et manipuler le client.
- Ajout du package *joueur.reseau* avec la classe *EchangeAvecLeServer* possédant les fonctionnalités d'échange avec le server.

Diagrammes de Séquences

Lancement et connexion d'un client :

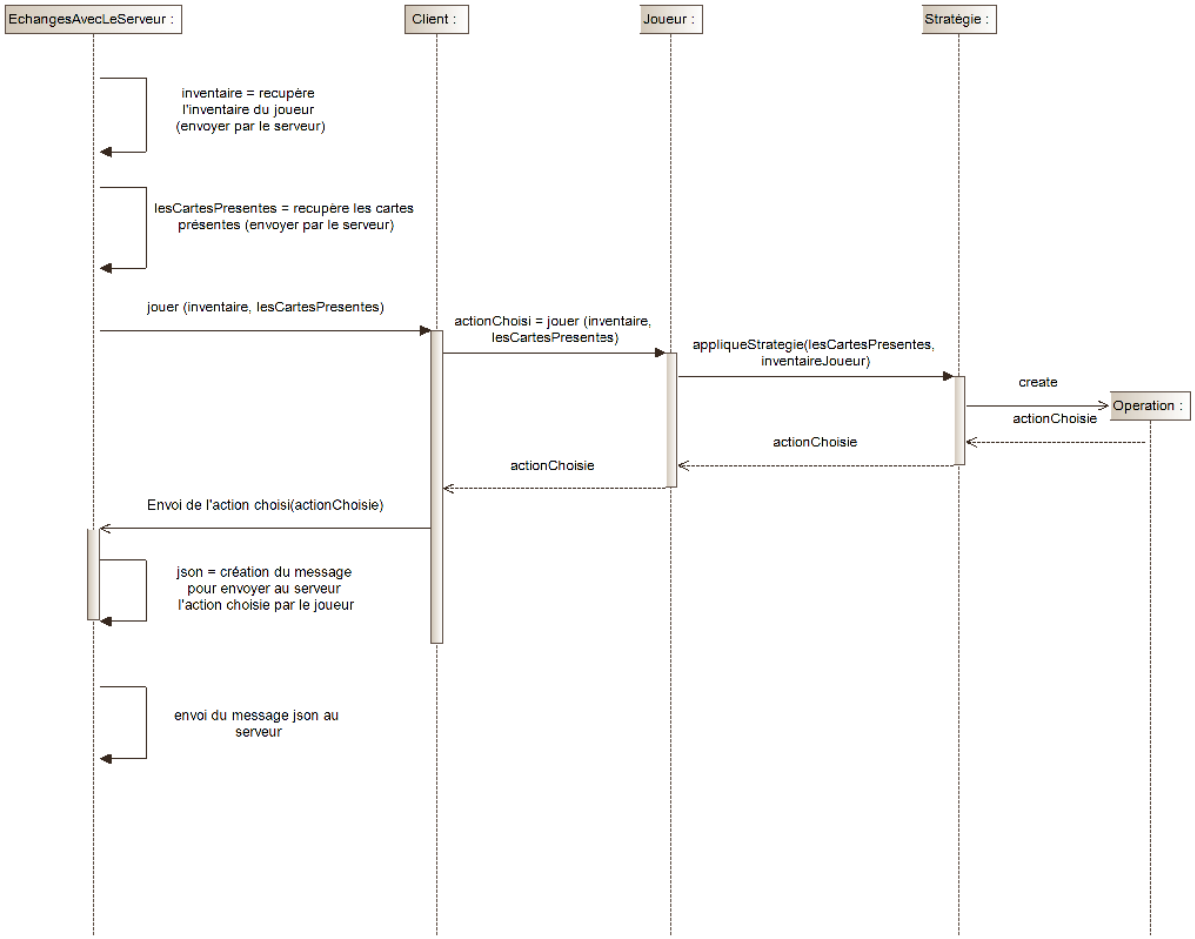


Use case : « se connecter »

Ce diagramme permet d'expliquer le lancement d'un client ainsi que sa connexion a un serveur.

Dans un premier temps, le Main analyse les arguments donnés par l'utilisateur (Mode de jeu, Affichage de la trace du jeu et des statistiques).
Dans un second temps, il génère des informations utiles pour la création d'un client, puis il le créer, et ce dernier vas créer un joueur.
Enfin, le Main demande au client de se connecte au serveur, et ce client se met en attente.
Le diagramme différent de la version intermédiaire, car dans la version précédente, la création des joueurs se faisait dans la main de lancement.
Or ici, nous devons exécuter 4 clients pour avoir 4 joueurs et ainsi commencer la partie.
De plus, les nom et stratégie des joueurs sont choisis de manière aléatoire alors qu'avant ils étaient écrits en dure dans le Main.

Jouer une action :



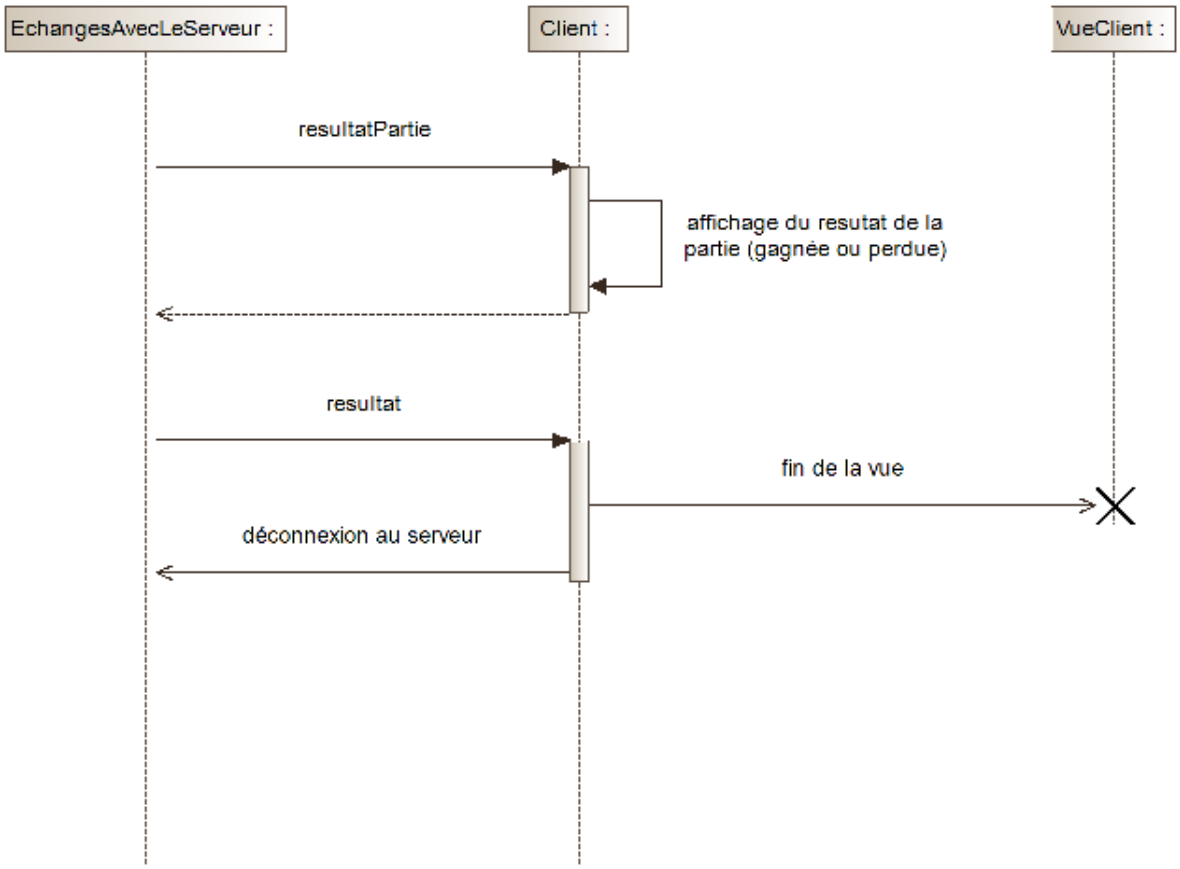
Use case : « Envoi de l'action à effectuer au server »

Nous détaillons ici le processus de demande de jouer et de choix d'action.

Tout d'abord, la partie réseau du client reçoit son inventaire et les cartes pouvant être piochés.
Grâce à cela, le client demande au joueur de choisir son action, et ce dernier applique sa stratégie qui renvoie une opération.
Cette opération est renvoyée au client et le client envoie à la partie réseau l'action à effectuer.
L'action est ensuite transformée en message json et est envoyée au serveur.

La différence ici est que nous avons ajouté un client.
Ce dernier fait l'intermédiaire entre la partie réseaux, qui envoie les demandes de jouer un tour, et le joueur, qui applique ça stratégie.
Ensuite, avant, le joueur était un bot qui choisissait lui-même l'action qu'il voulait faire.
Maintenant, pour choisir une action, le joueur doit appliquer sa stratégie (ce dernier lui étant attribué à sa création).

Fin d'une partie et déconnexion :



Use case : « Se déconnecter »

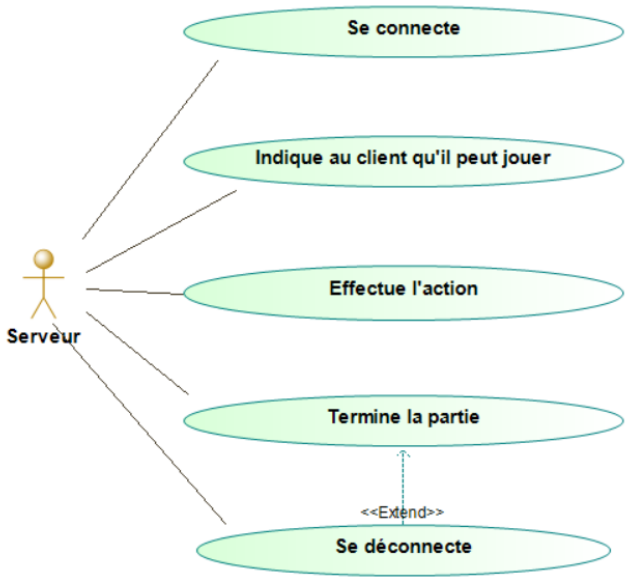
Ce diagramme montre comment un client se déconnecte lorsque l'on termine toutes les parties et que l'on se déconnecte.

Lorsqu'une partie se termine, le joueur est informé s'il a perdu ou gagné.
Si toutes les parties sont terminées, alors le client supprimera sa vue et se déconnectera juste après la réception du message.
À la différence de la version intermédiaire, le joueur n'a ni accès aux statistiques ni au nombre de points de tous les joueurs. Cependant, le serveur connaît ces informations.

Serveur / Moteur de jeu

Analyse des besoins

Diagramme de use case



Scénarios

Nom : Jouer
Acteur(s) : Serveur
Description : La creation d'une partie doit être possible pour un serveur
Auteur : Yannick ALCARAZ
Date(s) : 30/12/2020

Pré-conditions :
Un serveur doit être présent afin de pouvoir accueillir les clients et lancer la partie

Démarrage :
L'utilisateur a demandé de lancer le jeu

DESCRIPTION:
Le scénario nominal :

1. L'utilisateur ouvre un terminal au sein de l'IDE
2. L'utilisateur créer un serveur
3. Le serveur est en attente de client
4. Le serveur a accueilli les 4 clients et lance la partie
5. Le serveur indique au client qu'il peut jouer
6. Le serveur reçoit l'action que le client souhaite réaliser
7. Le serveur effectue l'action
8. Le serveur termine la partie, car un client a atteint le nombre de points de victoire à atteindre
9. Le serveur envoie le résultat de la partie à chaque client.
10. Le serveur s'arrête

Les scénarios alternatifs :
9a. Le serveur termine la partie et affiche les statistiques de la partie.

Post-conditions : Aucun

Conception logicielle

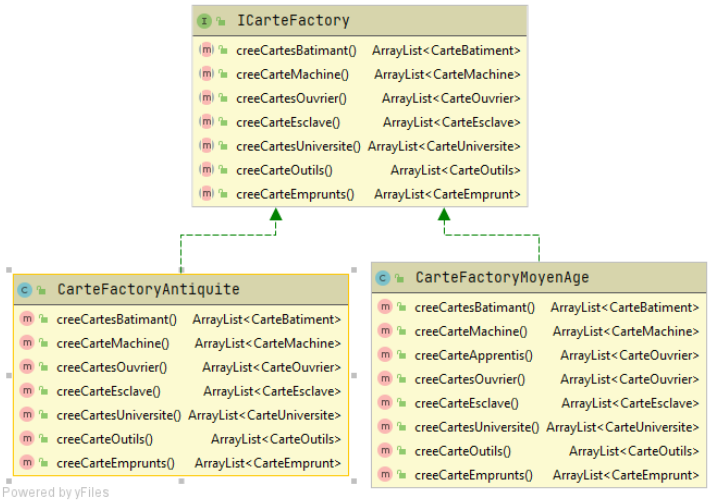
Diagrammes de Classes

Le module server permet de séparer les fonctionnalités exclusives du server afin que les modules commun et client ne puissent pas y accéder.

(Voir annexes, « Diagrammes modules, Dernière version » et « Diagrammes module Server, Générale »)

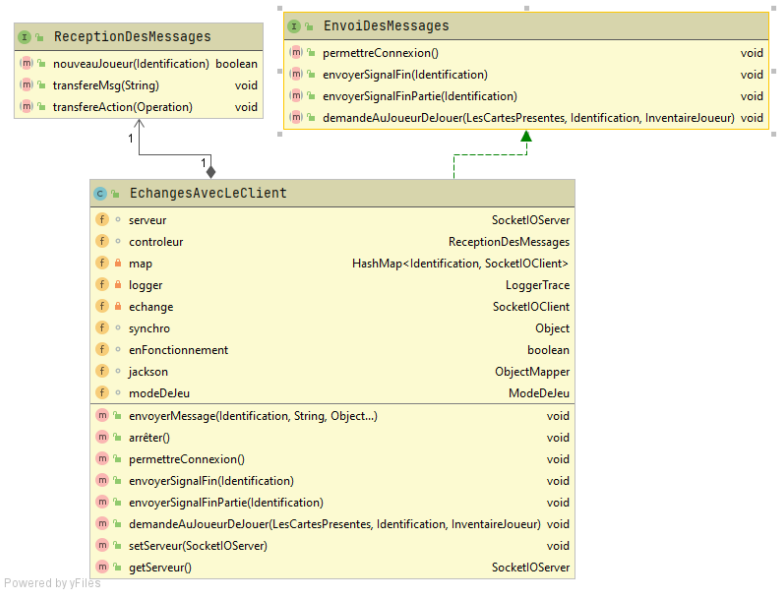
Package carte.factory

Package contenant des classes permettant de générer des collections de cartes selon le mode de jeu choisi.



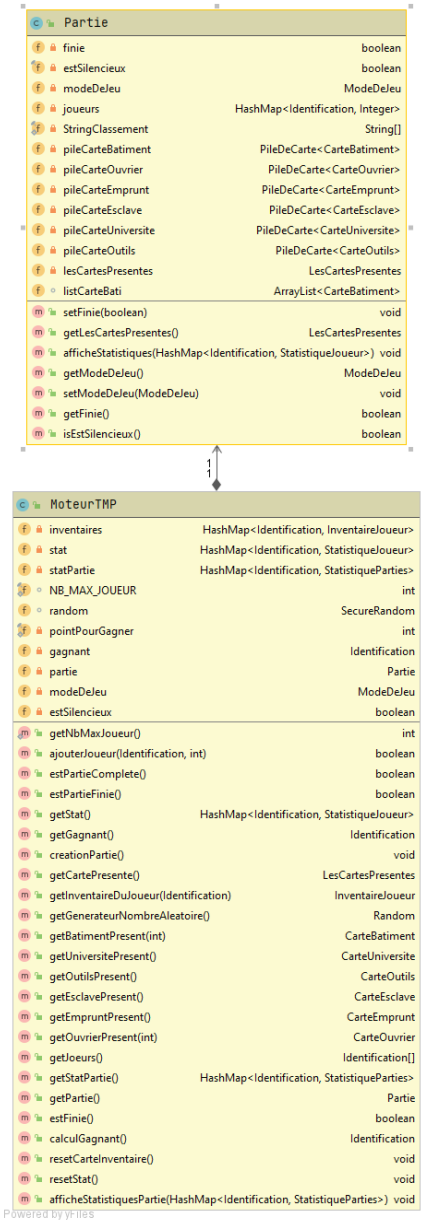
Package reseau

Package contenant 2 interfaces *ReceptionDesMessages* et *EnvoiDesMessages* qui permette d'échanger sont implémentés sur la classe *EchangesAvecLeClient* qui permet d'échanger avec les clients.



Package partie

Package contenant la classe *Partie* et la classe *MoteurTMP* qui permet de faire tourner une partie et de traiter les données nécessaires à la partie.

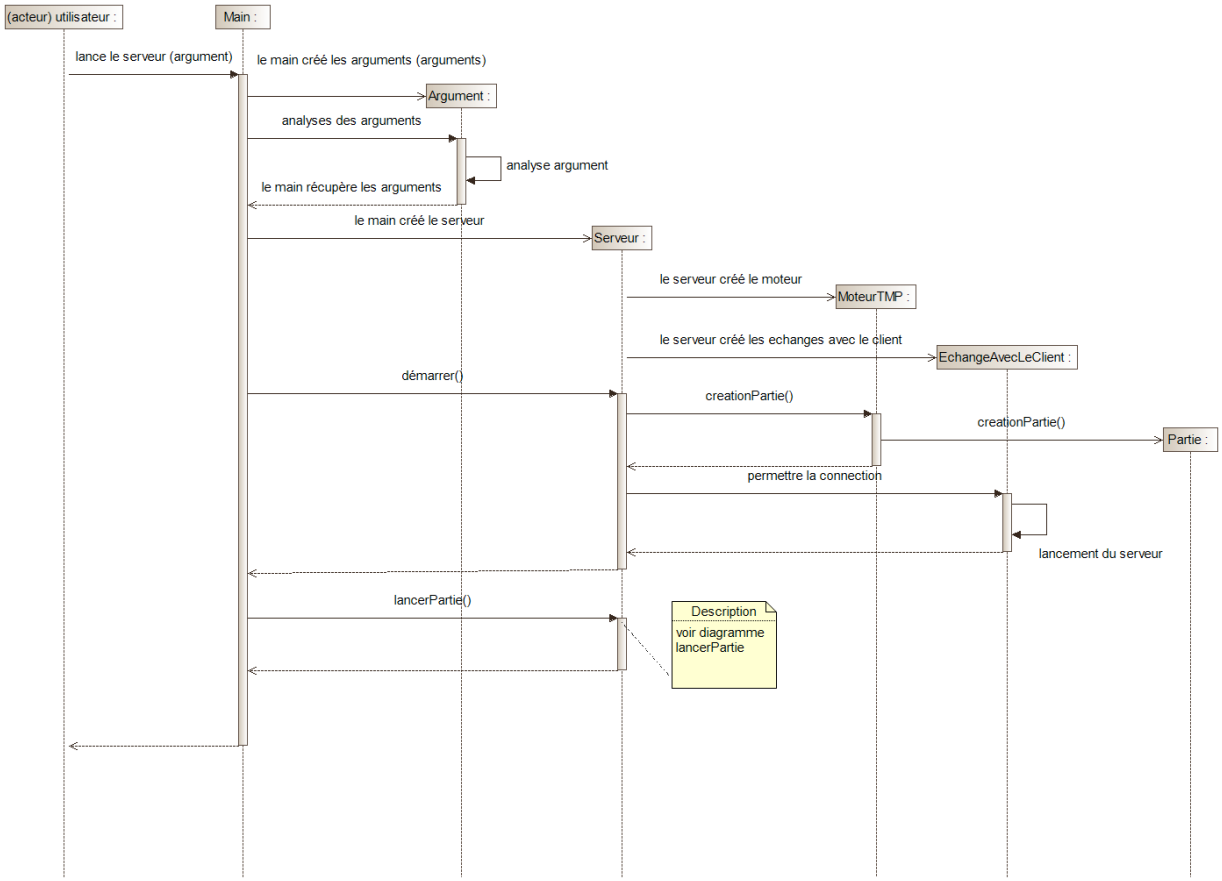


Autre Changement

- Ajout de la classe Server qui permet de paramétrer et manipuler le server.

Diagrammes de Séquences

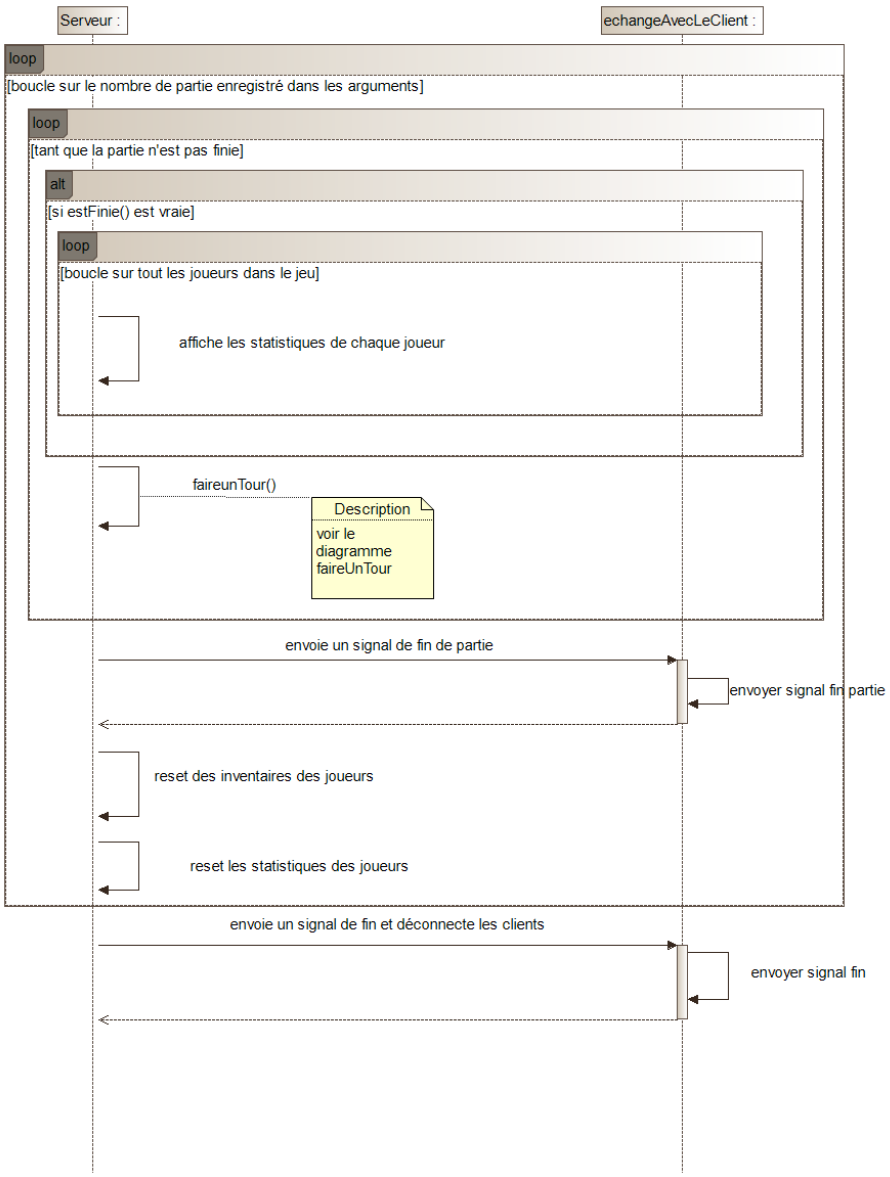
ConnectionServeur :



Use case : « Se connecter »

L'utilisateur lance le serveur avec des arguments sur la partie (nombre de partie, mode de jeu, affichage de la trace du jeu et des statistiques).
Le Main crée les arguments et crée un serveur. Le serveur créer le moteur, mais aussi les échanges avec le client.
Ensuite le Main démarre le serveur. Ensuite le serveur crée une partie, et pour finir le serveur se met en attente des clients.
Une fois tous les clients connectés la partie le lance en appelant la méthode lancerPartie.
Les arguments on était ajouté, le moteur n'existait pas et c'est le moteur qui crée une partie.

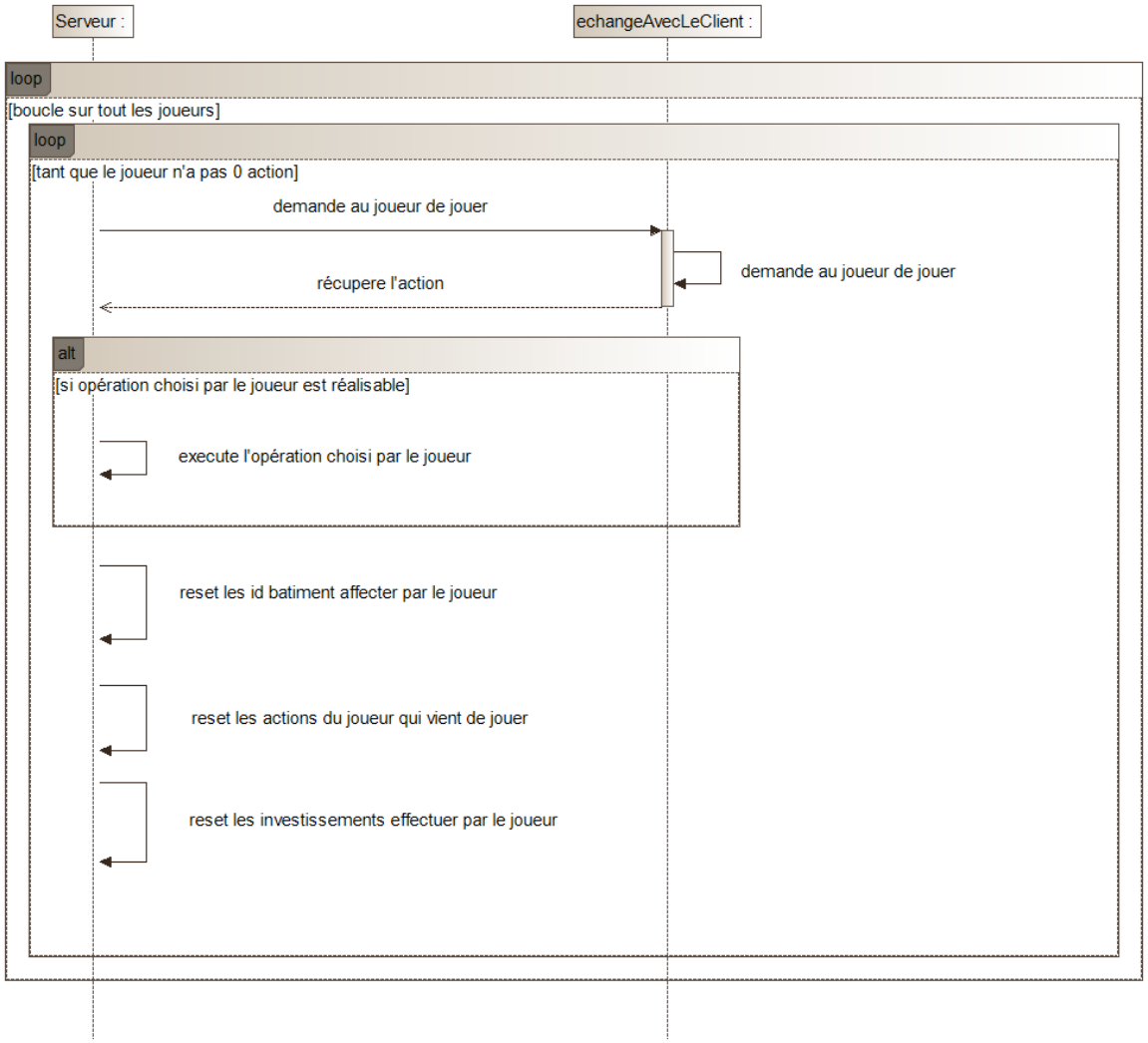
LancerPartie :



Use case : « Indiquer au client qu'il peut jouer », « Terminer la partie » et « Se déconnecte »

On boucle sur le nombre de parties enregistré dans les arguments. Tant que la partie n'est pas finie, on regarde si elle est finie.
Si elle est finie, on affiche les statistiques sinon on appelle la méthode faireUnTour.
Si la partie est finie alors on lance la méthode finPartie qui envoie un message au client et on reset les inventaires des joueurs et les stats des joueurs
Si la boucle de partie est finie alors on appelle la méthode fin qui met fin à la partie et déconnecte les clients.
Avant la partie demander au joueur de jouer via la méthode jouer qui bouclé. Maintenant c'est le serveur qui boucle sur l'identifiant des joueurs. Il demande au client de jouer, le serveur récupère l'action donner par le client, et l'exécuter.

faireUnTour :



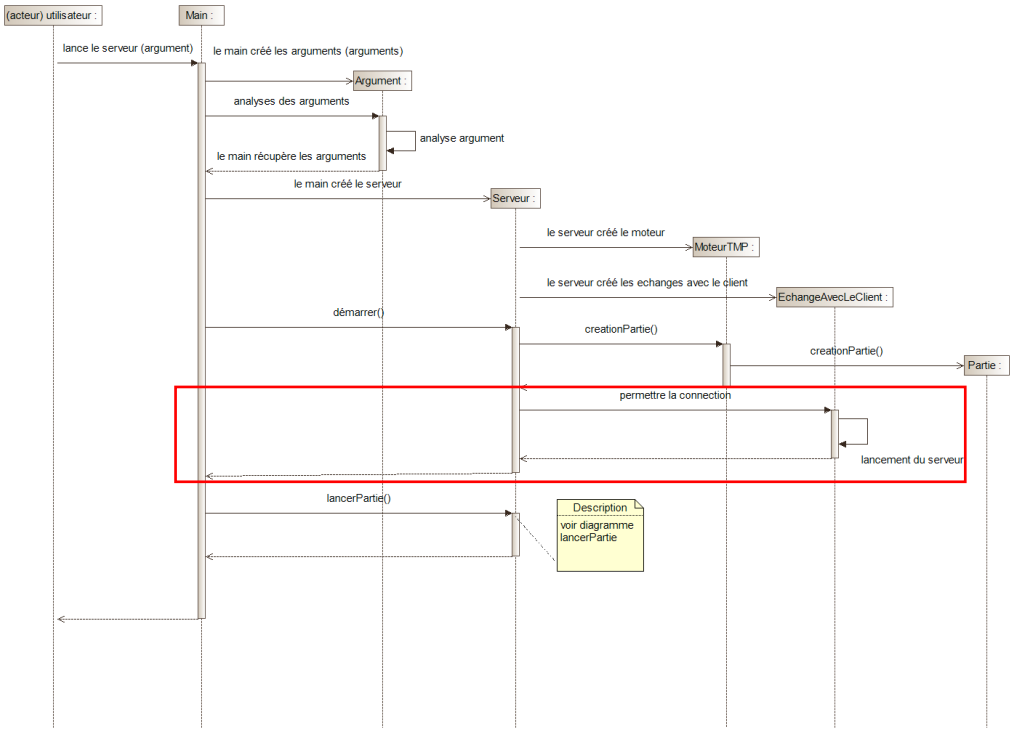
Use case : « Effectue l'action »

On boucle sur tous les joueurs, ensuite tant que le joueur qui joue n'a pas 0 action alors le serveur envoie une demande de jouer, ensuite le client envoie l'opération à faire.
Si l'opération est réalisable alors on exécute l'opération donnée par le client
Si le joueur a 0 action alors on lui reset les id bâtiment, les actions et aussi les investissements.
Ensuite on boucle sur le prochain joueur

Dans l'ancienne version la partie exécuter les actions. Dans la nouvelle version, c'est le serveur qui s'occupe d'exécuter les opérations.

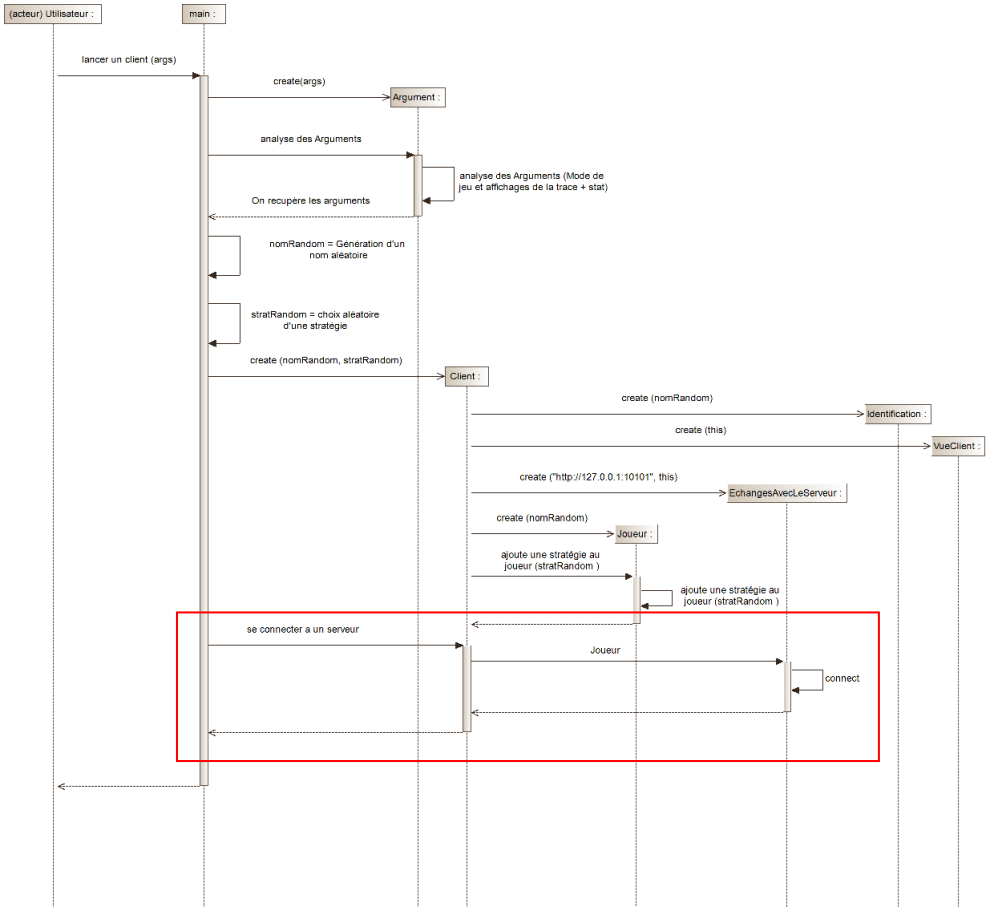
Interactions entre le ou les joueurs et le moteur

Connexion des clients au serveur :



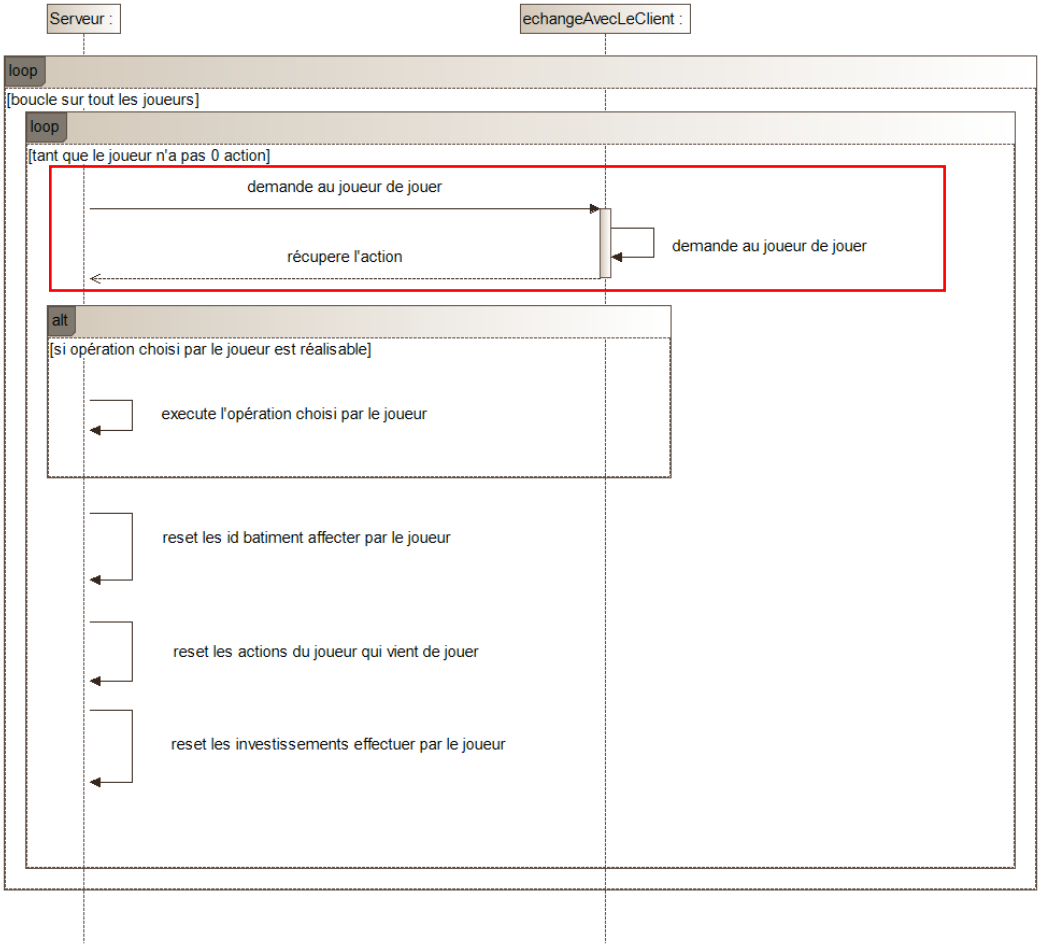
Le serveur permet au client de se connecter après la création de la partie.
Les messages concerné ‘IDENTIFICATION’ (message reçu) et ‘connect’ (message reçu)

Connexion d'un client à un serveur :



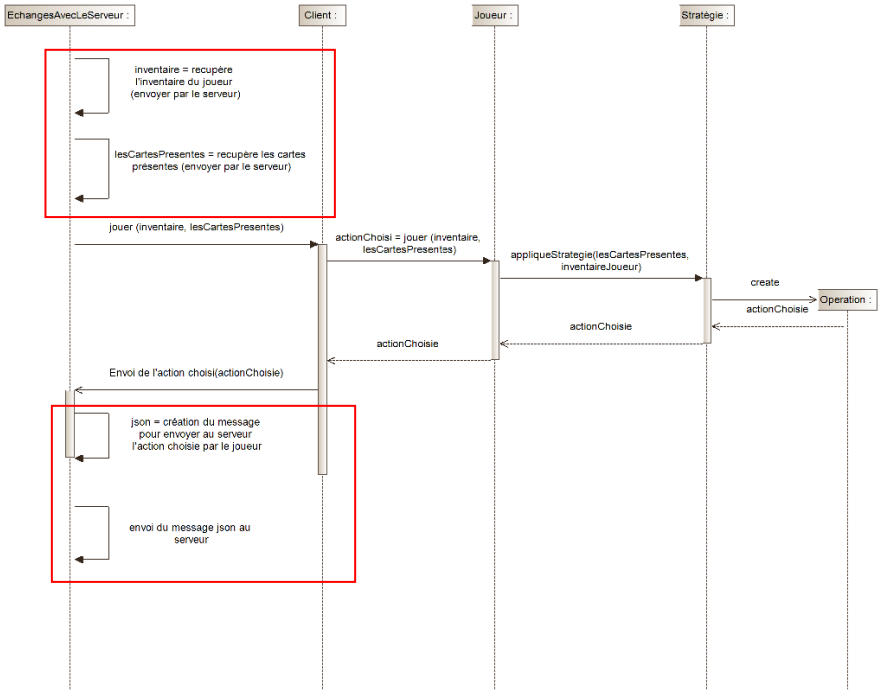
La demande de connexion se fait après la création du client.
Les messages concernés sont : 'connect' (message envoyé) et 'Identification' (message envoyé).

Demander et récupérer l'action :



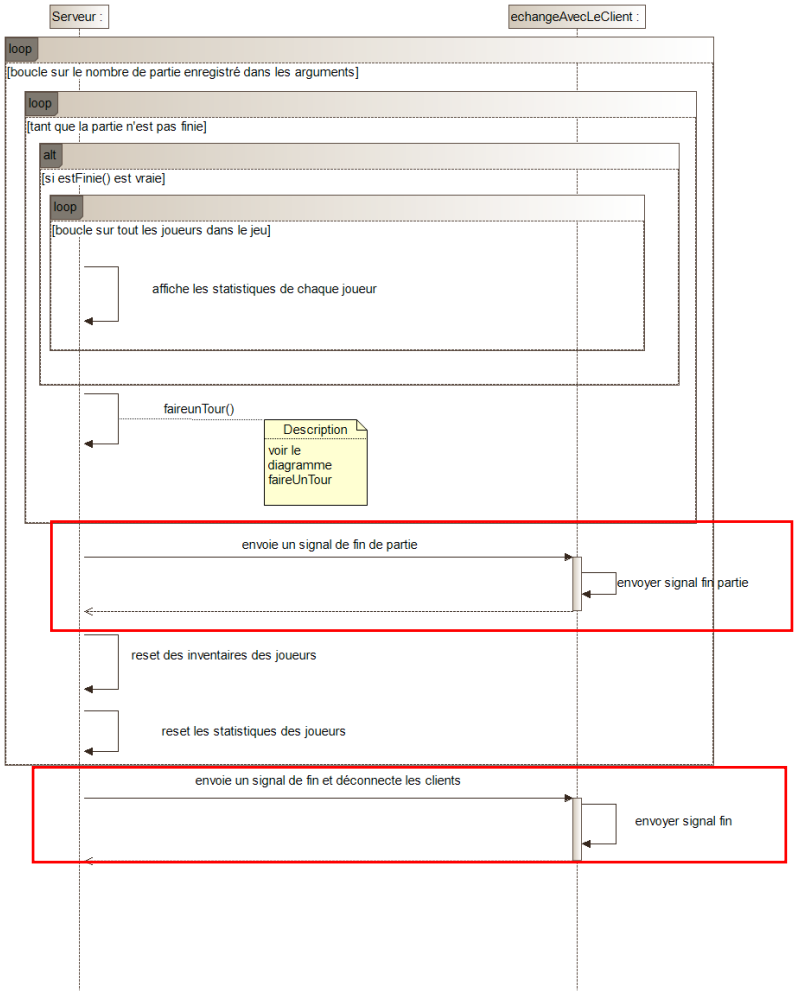
Le serveur demande au client de jouer et en retour il reçoit une action à faire.
Les messages concernés sont : 'DEMANDE_DE_JOUER' (message envoyé) et 'JOUER_CETTE_ACTION' (message reçu)

Demande de jouer :



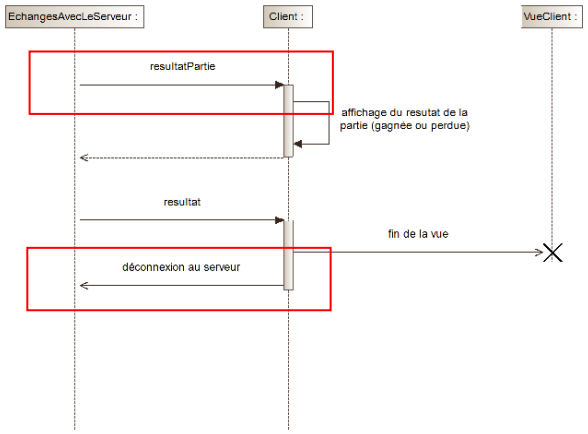
Le client reçoit une demande du serveur de jouer et vas alors récupérer son inventaire et les cartes présenter.
L'envoi de l'opération à effectuer au serveur est fait après que le jouer a appliqué sa stratégie.
Les messages concernés sont : 'Demande_De_Jouer (message reçu) et 'Jouer_Cette_Action' (message envoyé).

Fin Partie :



Lorsqu'on termine une partie, on envoie le résultat au client.
Quand toutes les parties sont finies, le serveur envoie l'évènement 'fin' au client pour dire que la partie est finie et que les clients doivent se déconnecter.
Les messages concernés sont 'FIN_Partie' (message envoyé pour chaque fin de partie) et 'FIN' (message envoyé pour la déconnexion)

Fin de partie :



Lorsqu'une partie se termine, le joueur reçoit le résultat de la partie.
Si toutes les parties sont terminées, il reçoit en plus un message de déconnexion.
Les messages concernés sont : 'Fin_De_Partie' (message reçu pour chaque fin de partie) et 'Fin' (message reçu pour la déconnexion).

Conclusion

Notre solution a en grande partie rempli les demandes du client. Cependant elle a tout de même des défauts et des points forts à souligner.
Vous retrouverez ci-dessous un tableau qui liste les points forts et les points faibles de notre projet.

Points forts	**Points faibles**
- Facilité dans l'ajout de nouvelles opérations et de cartes - La partie réseau et le reste du projet sont séparés - Nous avons des stratégies variées (faible, moyen et fort) - Respect des principes GRASP et SOLID - Code, en partie, sécurisé	- Utilisation difficile des cartes parfois - Le code avec le client/serveur est peu, voire pas tester - Les statistiques ne prennent pas en compte les investissements possibles - L'ajout de mode de jeu aurait pu être amélioré - Les opérations d'investissement ne fonctionnent pas, mais sont tout de même appelées (dans la branche Master) - Dues au point précédent, certaines stratégies peuvent boucler sur l'opération par défaut

Pour finir, nous parlerons de la suite du développement.

En tout premier lieu, il faudrait corriger le bug des opérations d'investissement.
Ensuite, je préconiserais l'ajout de test sur le serveur ainsi que le renommage de certaines méthodes et classes.
En troisièmes lieux, l'ajout de statistique sur les opérations d'investissement serait un gros plus ainsi que l'ajout des messages en couleur.
Enfin, il faudrait faire du refactoring dans le code (optimisation, passage à SonarLint du code, suppression de commentaire inutile).
Les points précédents sont pour nous les points essentiels.
Cependant si nous avions encore plus de temps, nous apprécierions l'ajout des fonctionnalités suivantes :

- Transmettre le mode de jeu du serveur au client,
- Envoi des statistiques de fin de parties du serveur au client (s'il le souhaite),
- Envoie du classement des joueurs et leur nombre de points en fin de partie,
- Améliorer de système de mode de jeu (créer une classe partie du moyen âge et partie de l'antiquité ?) ;

Glossaire

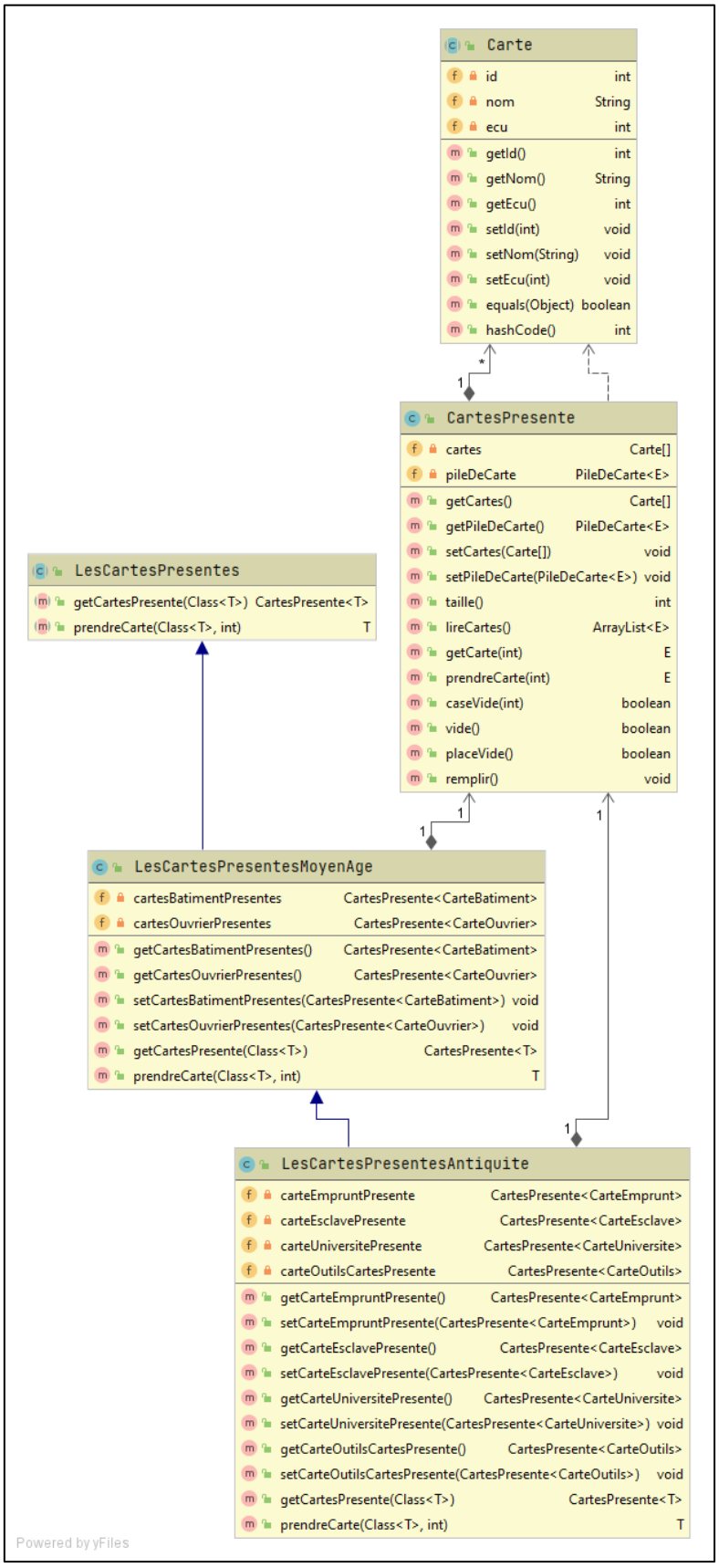
C

commanditaire
celui qui a commandé le logiciel 3

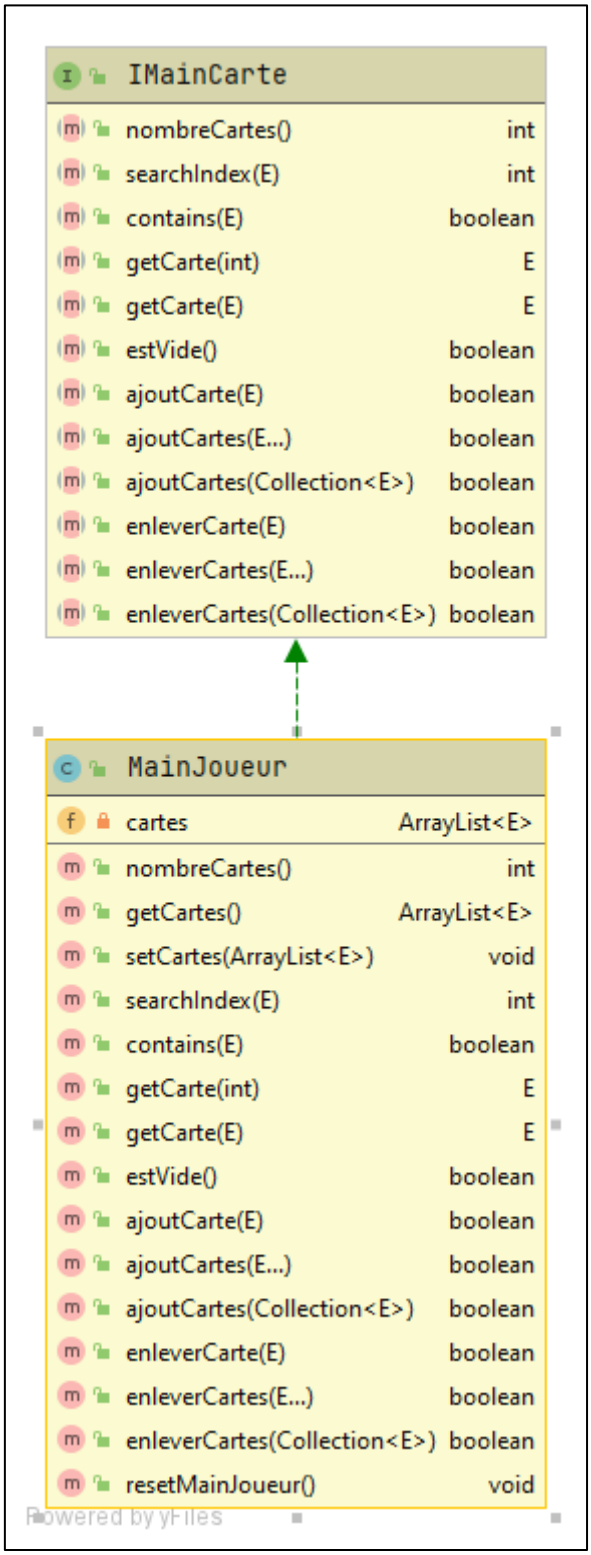
Annexe

Diagrammes module Commun

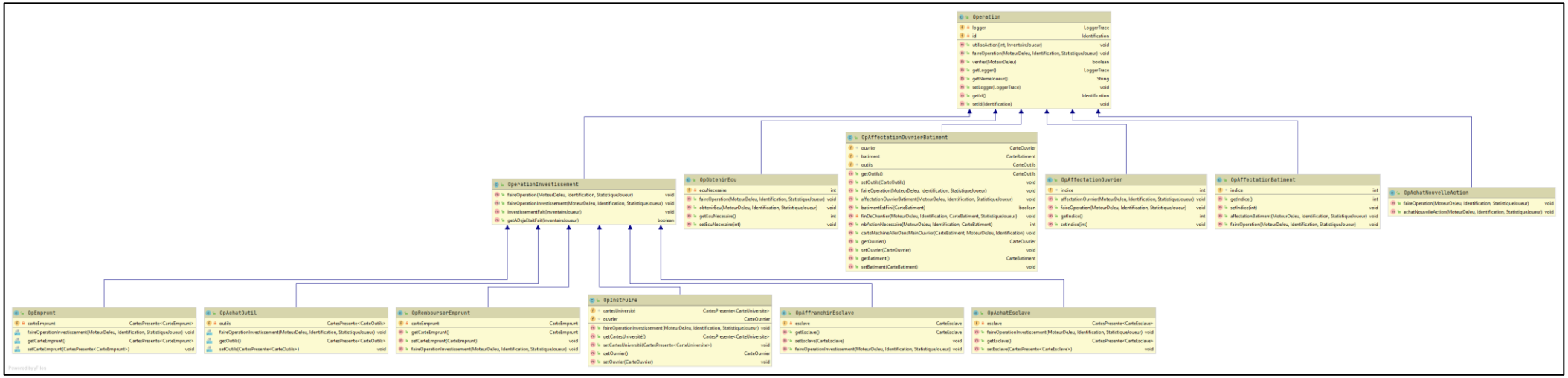
Package carte.cartesPresente



Package mainJoueur

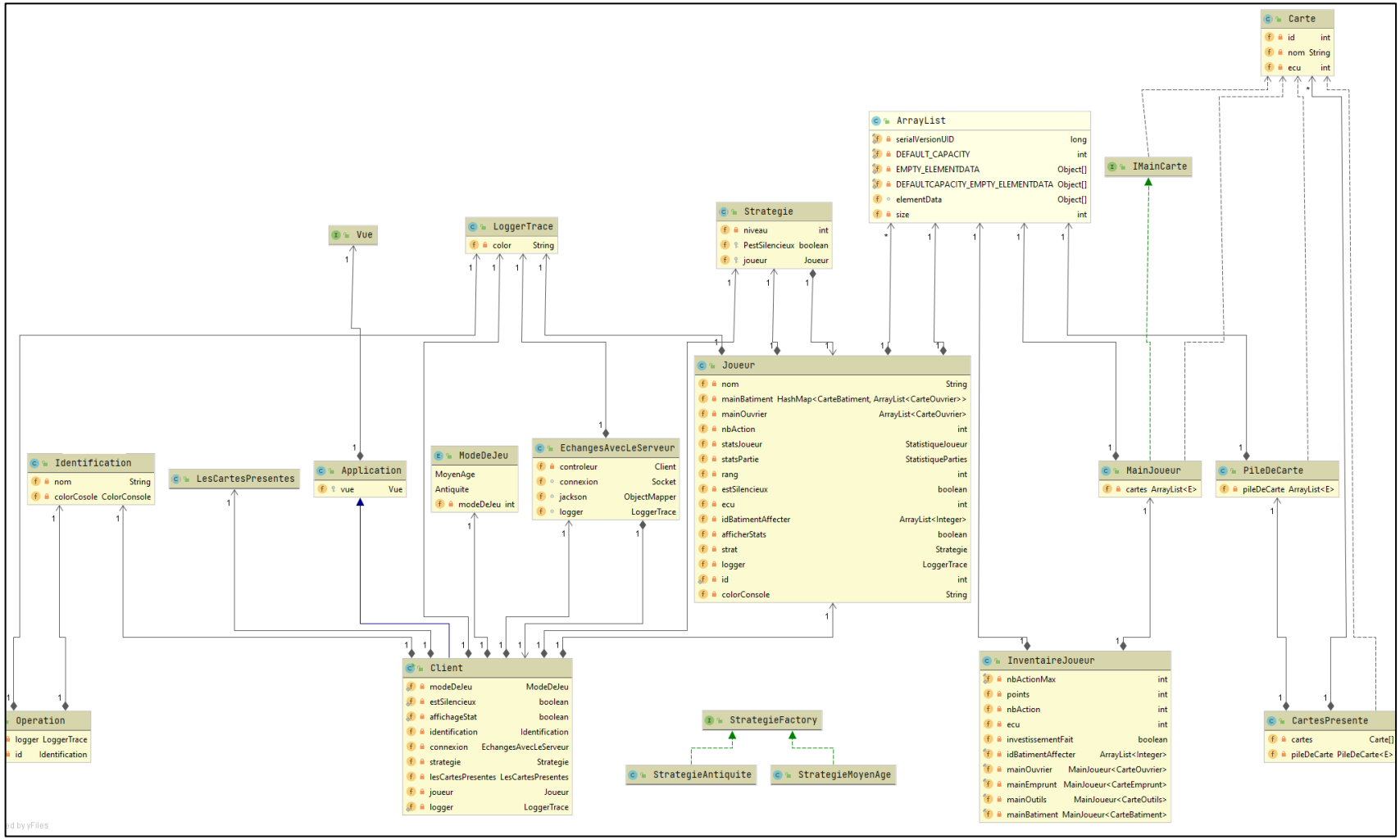


Package operation

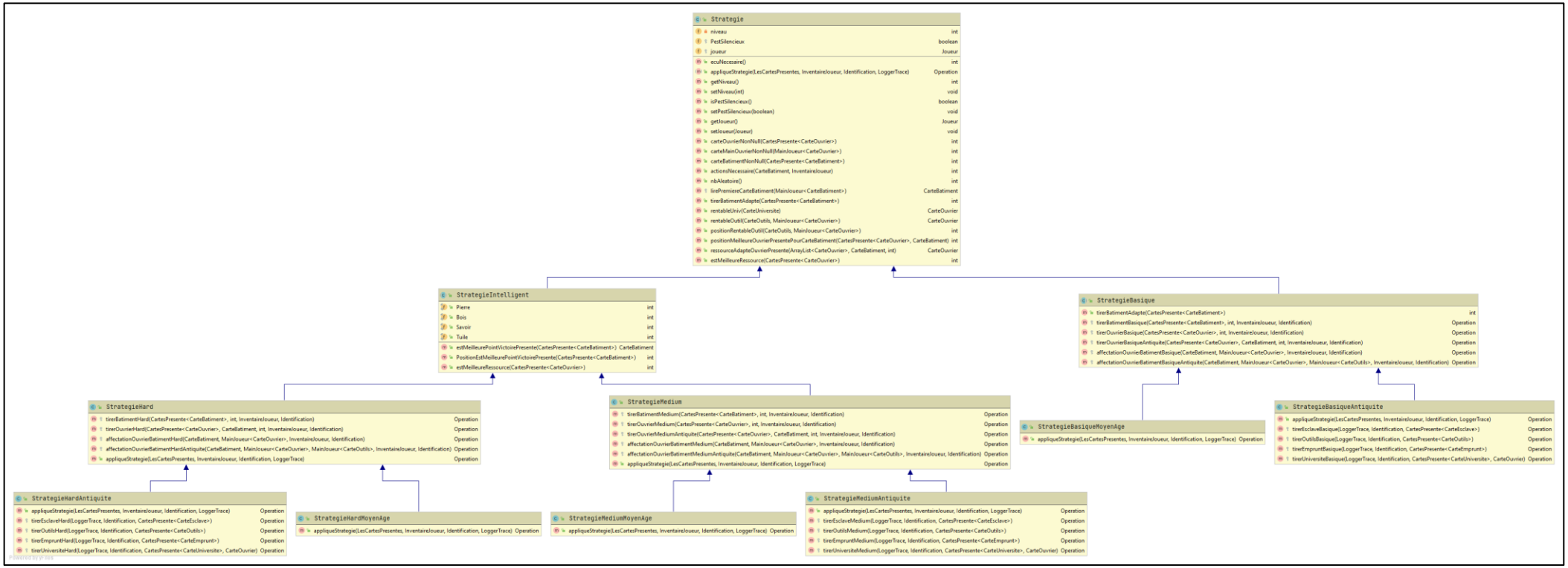


Diagrammes module Client

Générale



Package stratégie



Diagrammes modules Server

Générale