Projet de Développement

Conception Orientée Objet

UNIVERSITÉ DE NICE SOPHIA ANTIPOLIS

Mars 2020

Enseignant : Philippe RENEVIER

Licence 3 – MIAGE

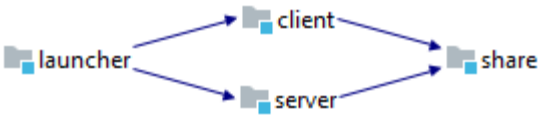
SESSION 2019 / 2020

Etudiants : BOUCHE Steven, CHAPOULIE Dorian, KAROUIA Alaedine, GARNIER Corentin, LONGIN Rémi

Table des matières

Architecture générale

Composition du projet Maven Diceforge



Le projet Diceforge est composé au total de 4 modules. Le module Launcher a pour rôle l'instanciation d'un serveur de jeu ainsi que l'instanciation des clients que devront se connecter au serveur sans l'aide du launcher en réseau.

Le module serveur lui-même créer la partie réseau via socketio netty et héberge toutes les données de la partie dans les classes dédiées dans le module share. Le module share est composé de toutes les classes utilisables par le client et le serveur permettant d'éviter des dépendances cycliques.

Le module client lui aura simplement pour rôle de se connecter et d'attendre des demandes serveur, faire un choix en fonction des données passer par le serveur dans la demande, puis d'envoyer sa réponse au serveur. Plus tard il serait intéressant pour chaque action d'un client de broadcast le résultat de l'action à tous les clients afin que l'IA puisse prendre en compte toutes les données de la partie. Pour le moment le client choisit de façon aléatoire parmi les différents choix, il n'a donc pas besoin de toutes les données du plateau pour l'instant.

Exemple de classe en fonction des packages :

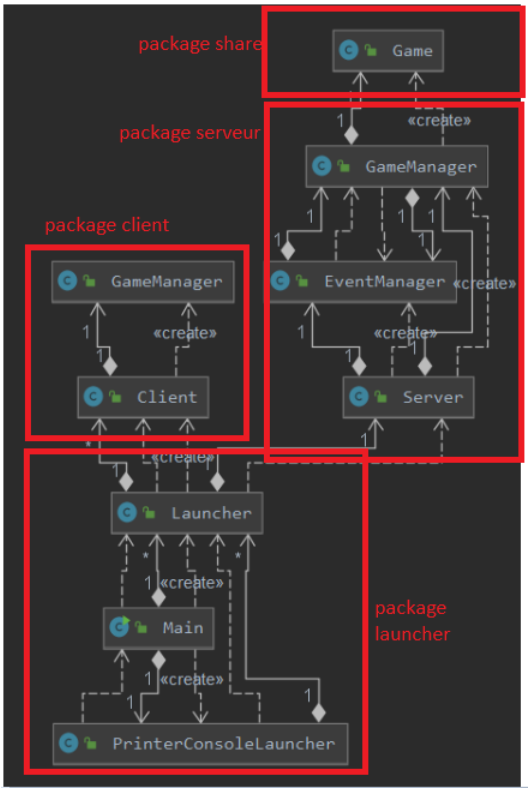


Diagramme d’activités

Notre projet est composé de plusieurs activité. L’activité Main qui lance l’activité du launcher qui lui-même lance le serveur de partie ainsi que ses clients. Une fois tous les clients connecter le serveur exécute la partie. Dans cette partie je ne détaillerais pas l’activité concernant l’exécution de la partie car plus détailler dans la partie du serveur.

Diagramme activité Main

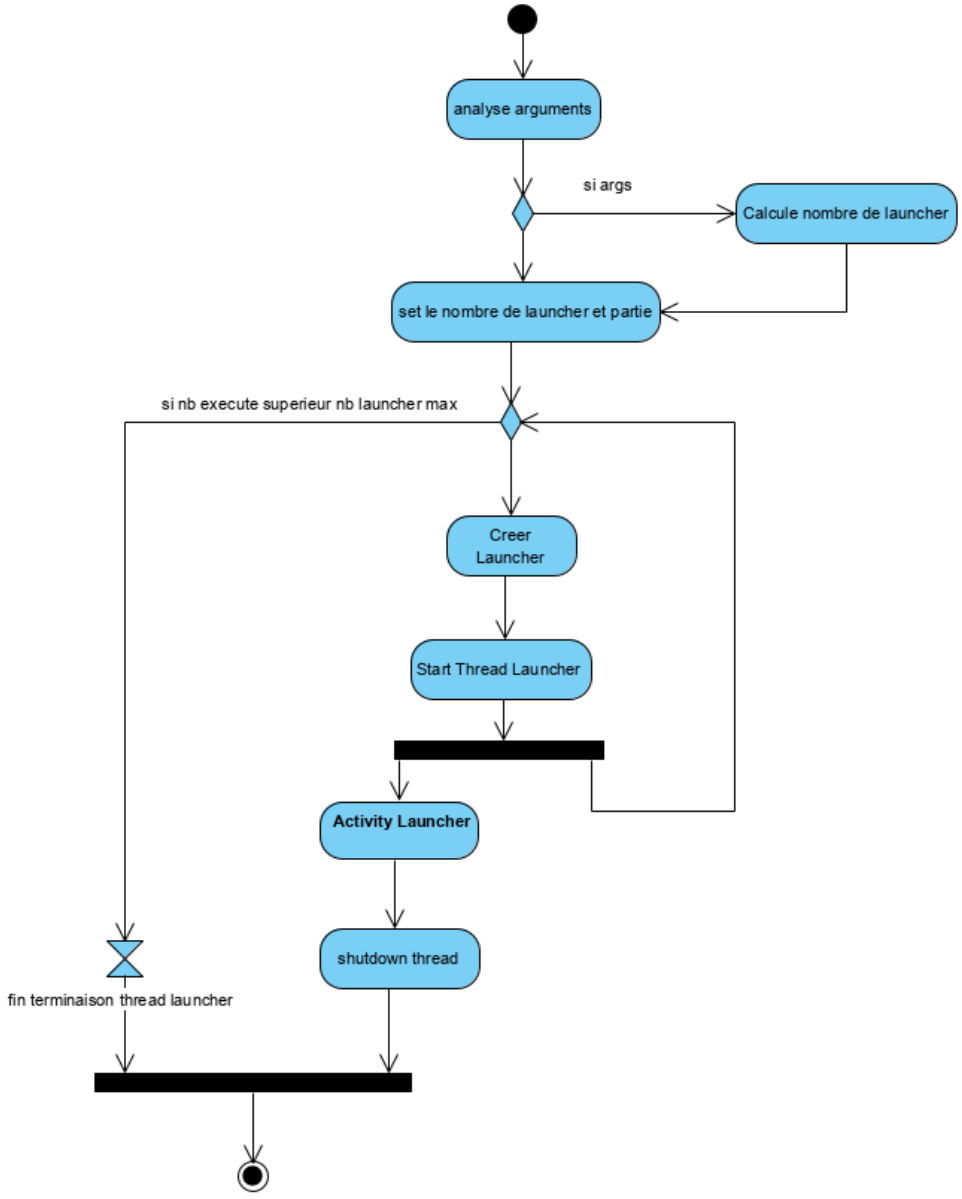


Diagramme activité Launcher

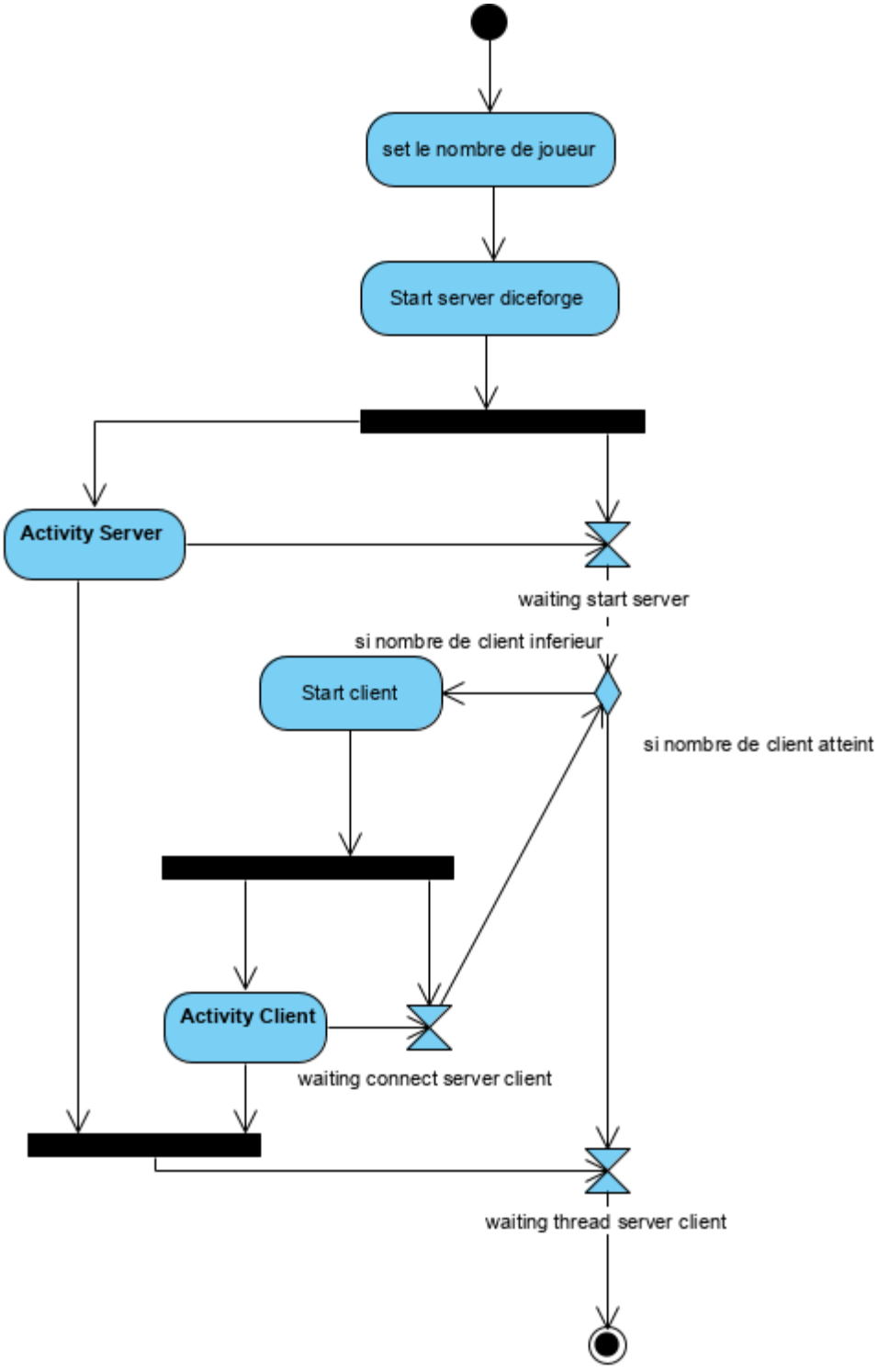
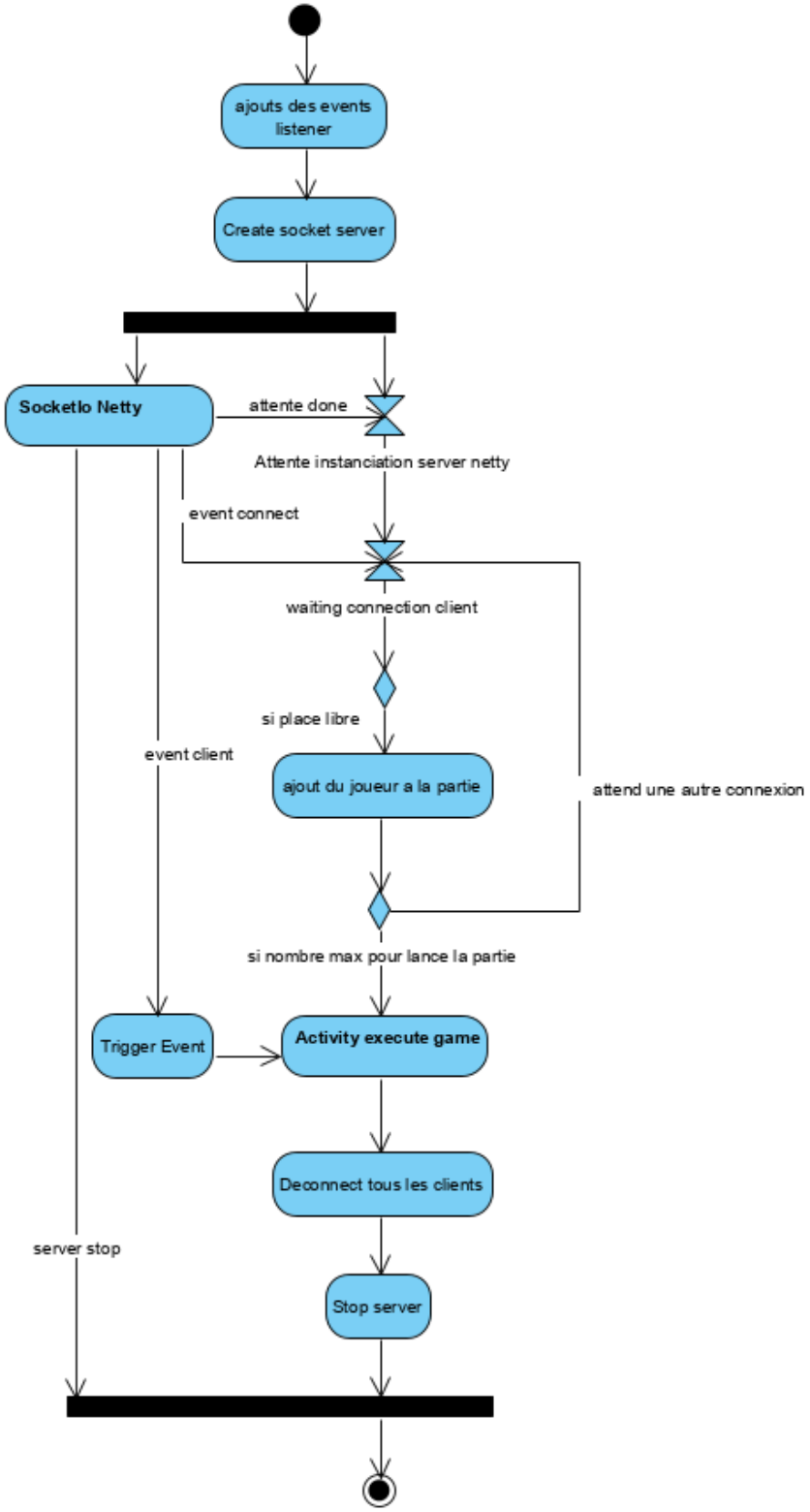


Diagramme activité Serveur



Fonctionnalités

Le programme est composé de deux modes. S'il n'y a pas d'arguments alors il lance un launcher qui exécutera une partie de DiceForge. Le launcher lui lance un server de partie ainsi que tous les clients nécessaires afin de lancer une partie, actuellement uniquement la version 4 joueurs du jeu.

La partie s'exécute alors en distribuant l'argent de début de partie à tous les joueurs en fonction de leur ordre d'arrivée. Puis déclenche une faveur majeure pour tous les joueurs. Si le joueur tombe sur une face simplement avec un gain de ressource alors l'ajoute dans son inventaire sinon lui demande de choisir la ressource voulue pour les faces à choix multiple. Une fois que tous les joueurs ont répondu il demande au joueur actif qu'il a au préalable sélectionner s'il veut forger une face sur un de ses dés ou acheter une carte. Une fois son choix fait il répond au serveur avec sa réponse et le serveur exécute le choix du client.

Enfin il met fin au tour actif. Une manche est composée de tour actif étant égal au nombre de joueur. Une partie est composé de 9 manches.

Dans le cas où l'argument "-p" suivie d'un nombre de partie est spécifié le programme divisera le nombre de partie à exécuter en fonction d'une taille de pool de launcher dans une classe de configuration. Il lancerait alors X launchers avec chacun Y parties à exécuter. Le programme s'arrête quand tous les launchers ont fini leur job.

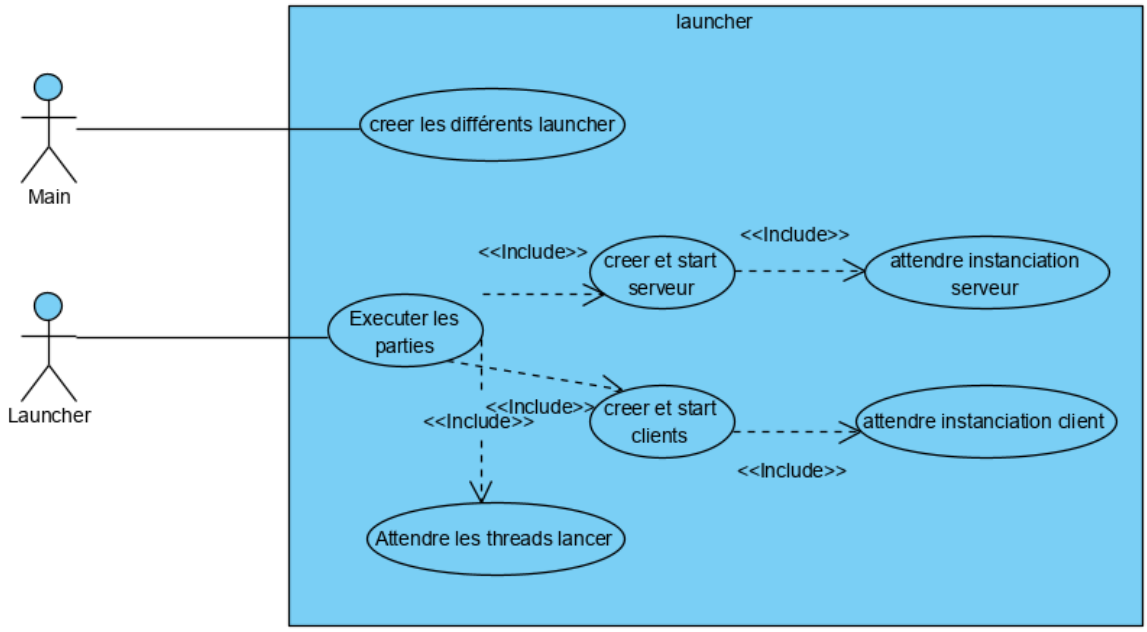
Fonctionnées qui sont implémentés :

- Client / Server
- Boucle de jeux
- Forge
- Temple
- Face Simple de dés
- Face hybride de dés
- Iles
- Cartes sans actions
- Action des faces hybrides
- Action des faces simples
- Multiparti

Module principale (main et launcher)

Analyse des besoins

Use case



User story

En tant que Launcher je souhaite pouvoir exécuter X partie

En tant que Launcher je souhaite pouvoir créer le thread serveur et le start

En tant que Launcher je souhaite pouvoir créer les threads clients et les starts

En tant que Launcher je souhaite pouvoir attendre la fin de la partie

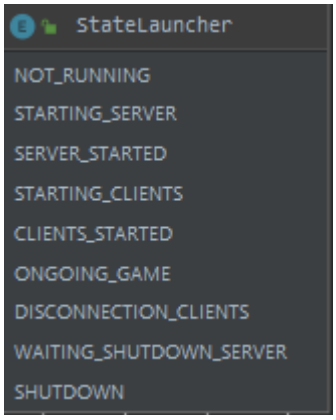
Scénario

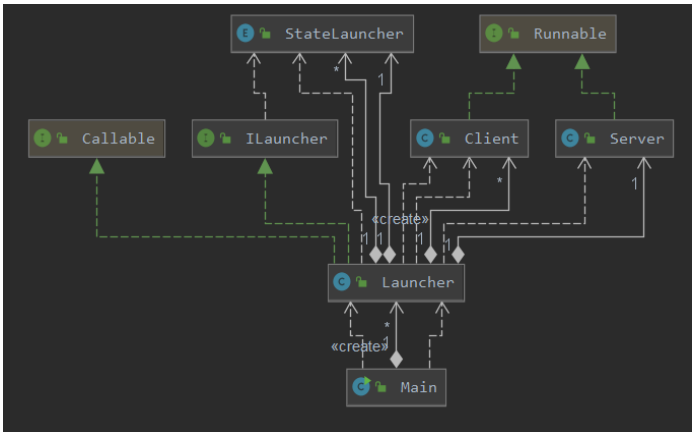
1. Créer et lancer serveur
    a. Instancie un objet serveur avec un port particulier
    b. Lance le server dans un thread
    c. Attend l'instanciation du server
    d. Instancie les clients avec le port serveur
    e. Start les clients dans plusieurs threads
    f. Attend la fin d'exécution du serveur et des clients

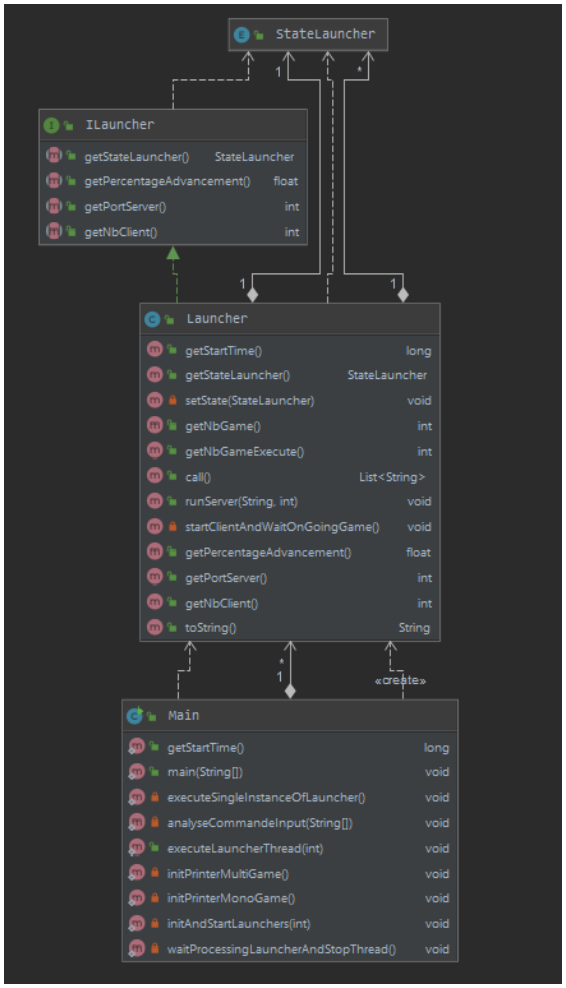
2. Créer et lancer Client
    a. Instancie un objet client avec tous les paramètres
    b. Lance le client dans un thread
    c. Attend la notification du client qu'il c'est bien connecter
    d. Si le nombre de joueur actuel est inferieur joueur max alors retourne à l'étape a sinon attend la fin des threads clients

Conception logiciel

Point de vue statique





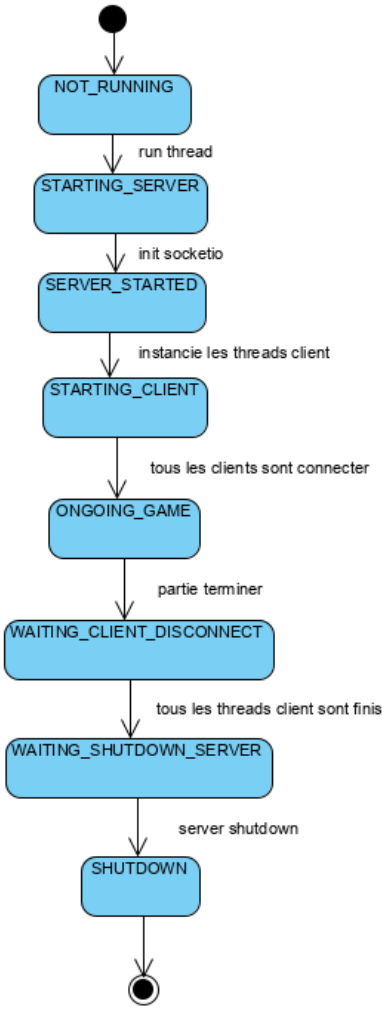


Le launcher est instancier par le programme principal. Le launcher doit exécuter un certain nombre de partie. On peut alors voir sur le schéma au-dessus que launcher lance X thread clients ainsi qu'un thread serveur des autres modules.

Le launcher à un état qui évolue au cours du temps dans l'enum StateLauncher. Les transitions entre les états sont décris en dessous via le diagramme d'état launcher.

Point de vue dynamique

State diagramme Launcher

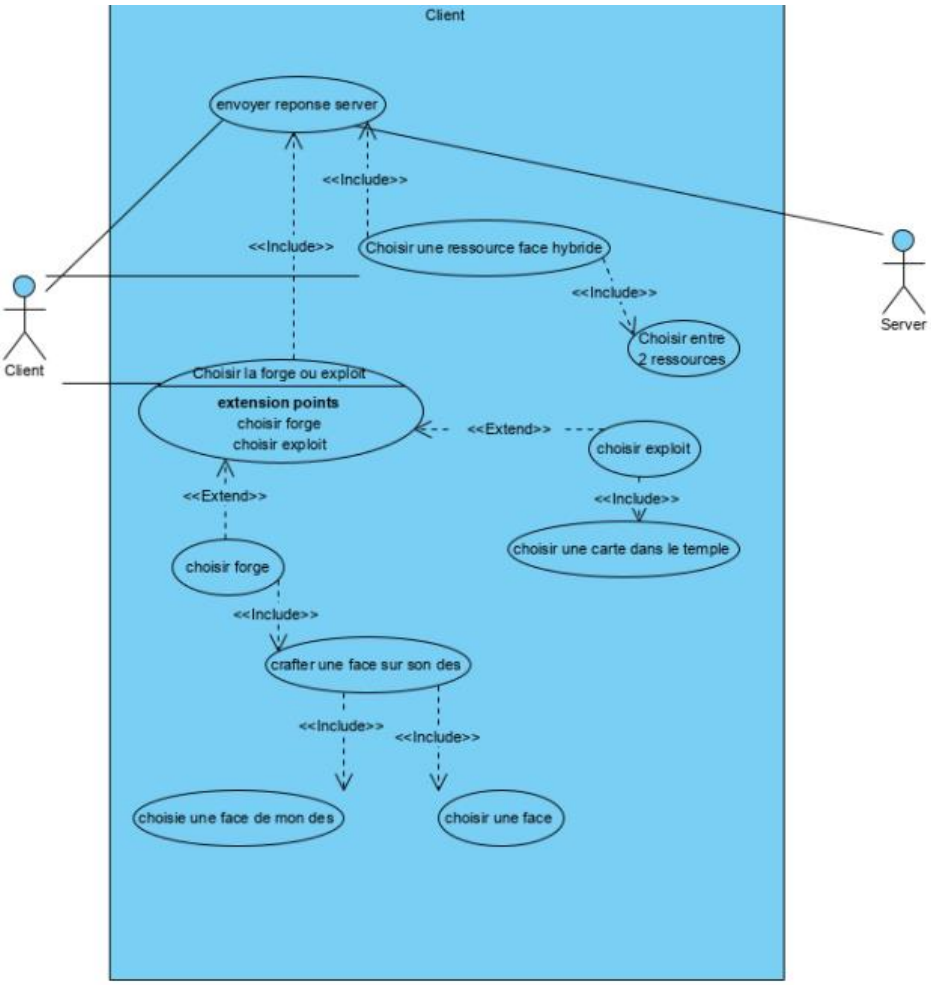


Module Client

Analyse des besoins

Les fonctionnalités réseau seront exposé dans la partie client/server plus en détail.

Use case



User story

En tant que client je souhaite pouvoir choisir si je veux forger une sur mon dé ou acheter une carte.

En tant que client je souhaite pouvoir choisir entre 2 ressources si je roll une face hybride

Scénario

1.choixForgeorExploitorNothing

a. choisirForge si je peux

b. crafter une face sur son dé

c. choisir le dé

d. choisir la face à modifier

e. sinon choisirExploit si possible

f.choisir une carte dans le temple

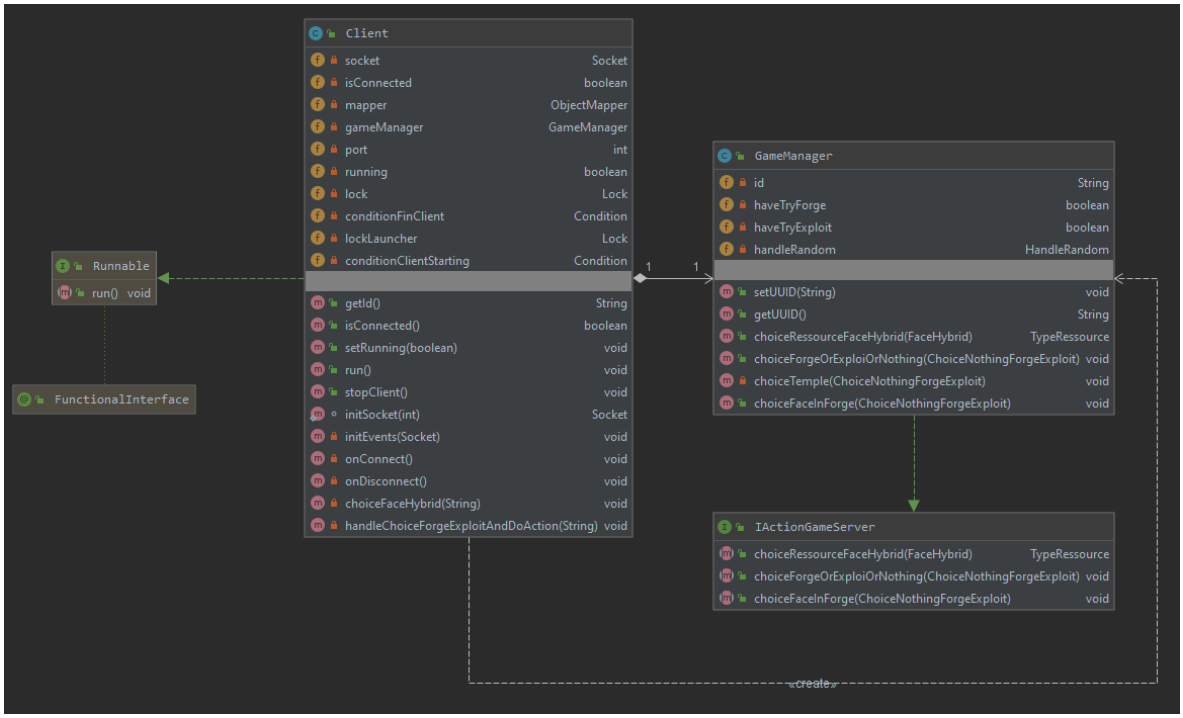
g.sinon Nothing

2.choixFaceHybrid

a. choisir carte dans le temple

Conception logiciel

Point de vue statique



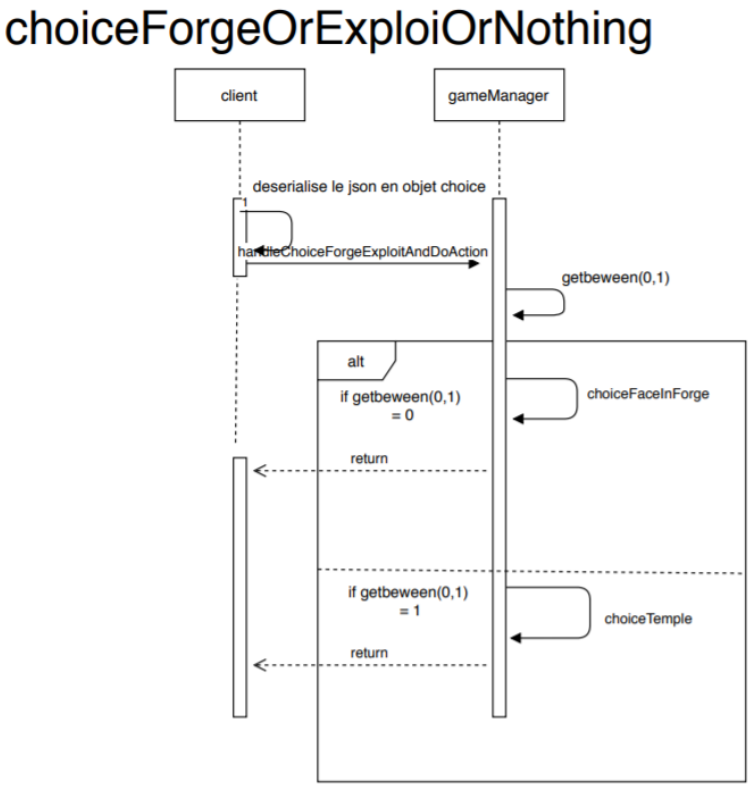
Le client est composé d'un gameManager. Le gameManager à pour rôle d'effectuer les décisions du choisir en fonction de l'objet envoyer par le serveur. Il contient alors les méthodes pour choisir de craft une face sur le dé ou alors d'acheter une carte en fonction de l'event que le serveur lui envoi. Tous l'aspect réseau est traiter dans la partie réseau.

Point de vue dynamique

ChoiceForgeOrExploiOrNothing

Précondition : Le serveur a précédemment envoyé une demande de choix.

Postcondition : Le client a renvoyé l'objet choice rempli avec ses décisions.



Le client désérialise le JSON en objet choice envoyé précédemment par le serveur puis ensuite l'envoie au gameManager qui possède une méthode public pour gérer le choix et des méthodes privé pour compléter l'objet choice en fonction du getbetween(0,1).

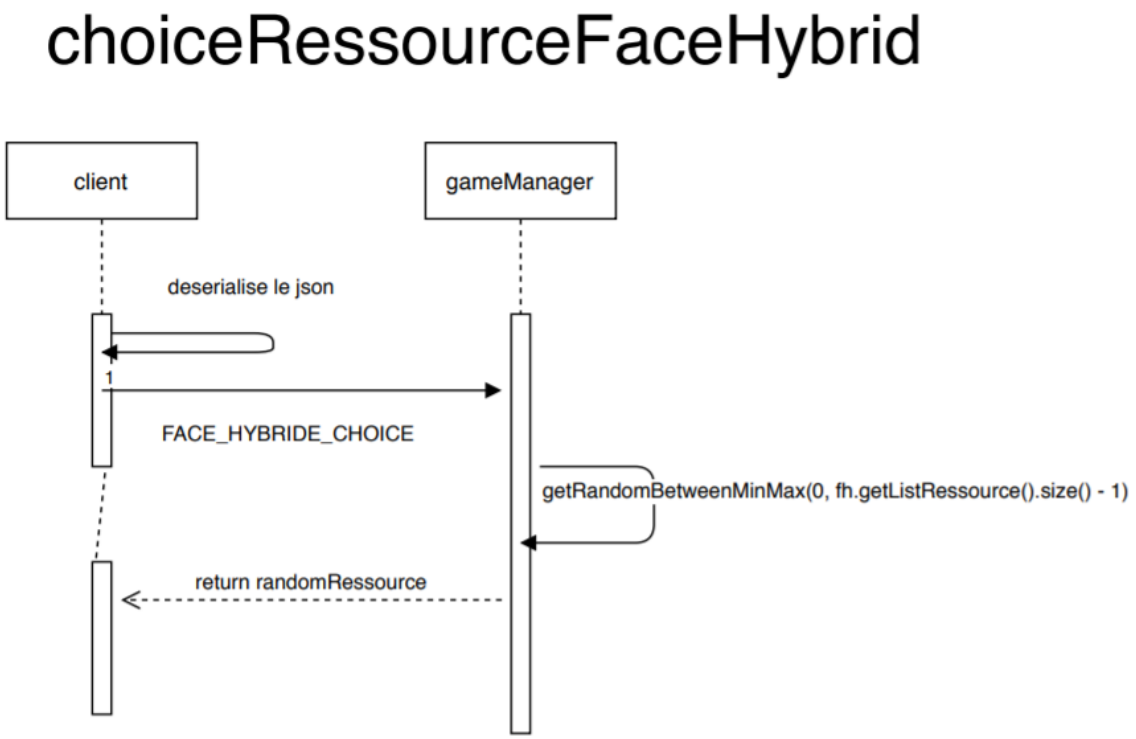
S'il obtient un 0, il utilise la méthode privé choiceFaceforge et complète l'objet puis return.

S'il obtient un 1, il utilise la méthode privé choiceTemple et complète l'objet puis return.

ChoiceRessourceFaceHybrid

Précondition : Le serveur a précédemment envoyé une demande de choix.

Postcondition : Le client a renvoyé la ressource aléatoire.



Le client désérialise le json renvoyé par le serveur puis envoie envoie au gameManager FaceHybrideCHOICE.

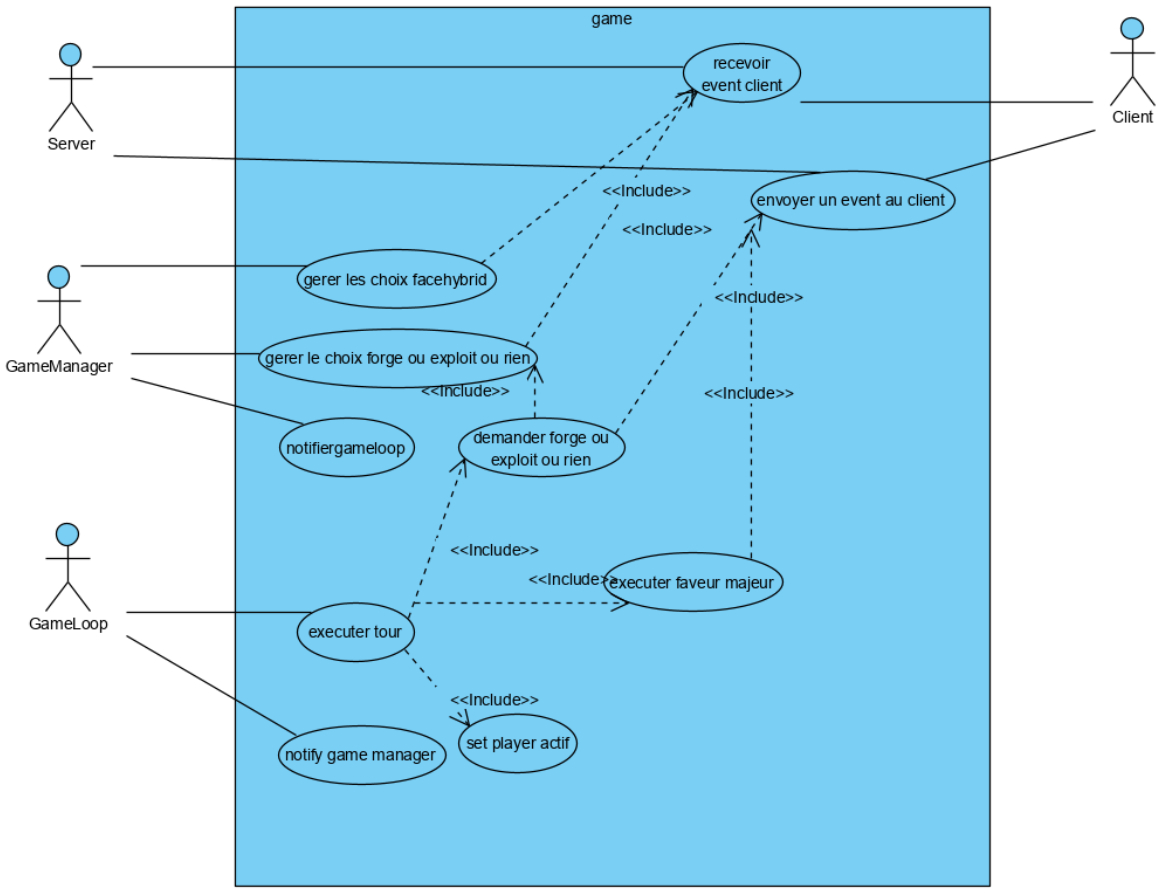
En fonction du random, une ressource va être choisi puis renvoyé au client.

Module Serveur

Analyse des besoins

Les fonctionnalités réseau seront exposé dans la partie client/server plus en détail.

Use case



User story

En tant que serveur je souhaite pouvoir exécuter une partie

En tant que serveur je souhaite pouvoir connecter des clients

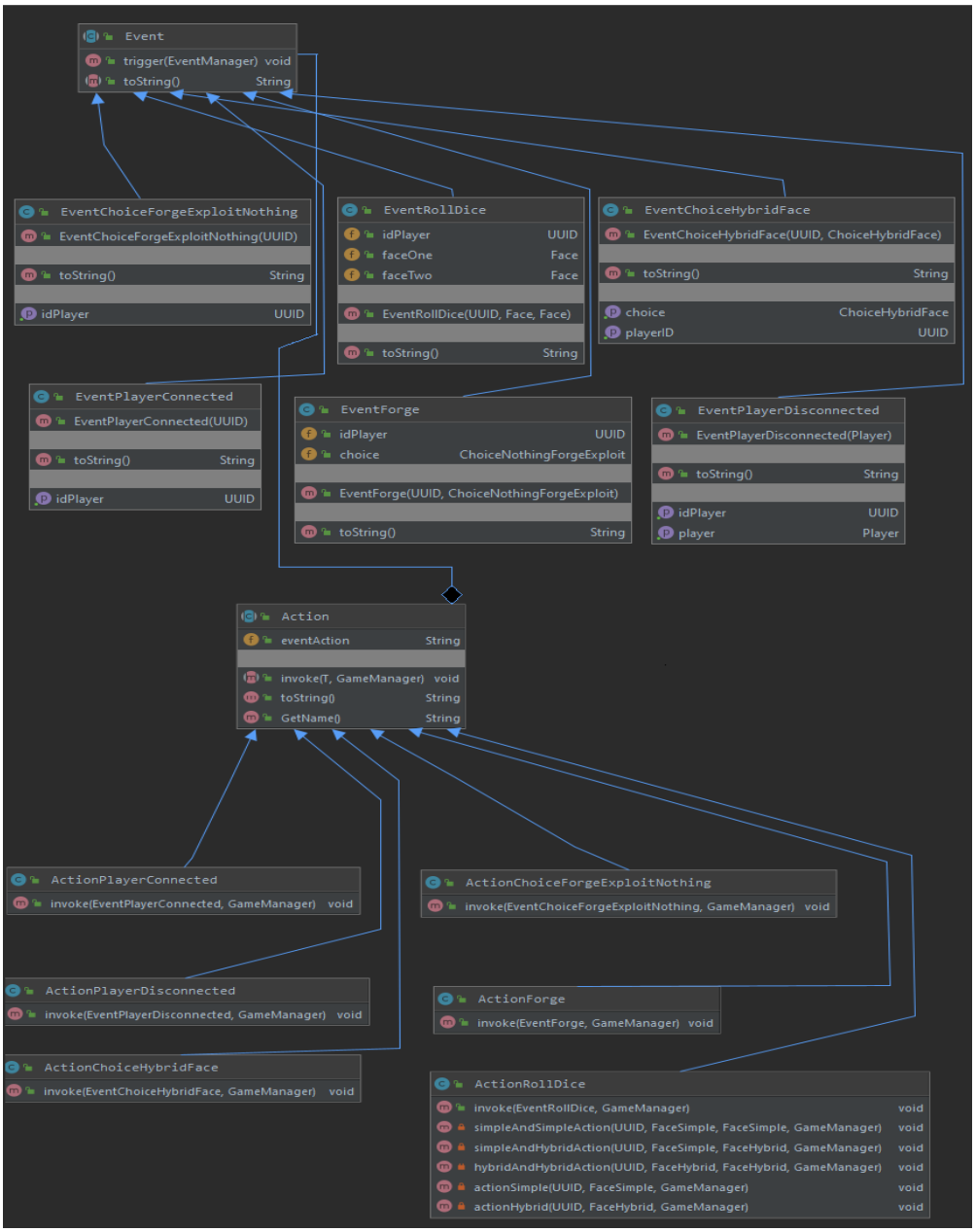
En tant que serveur je souhaite pouvoir gérer les différents event client

- Choix face hybrid
- Choix forge ou temple

Conception logiciel

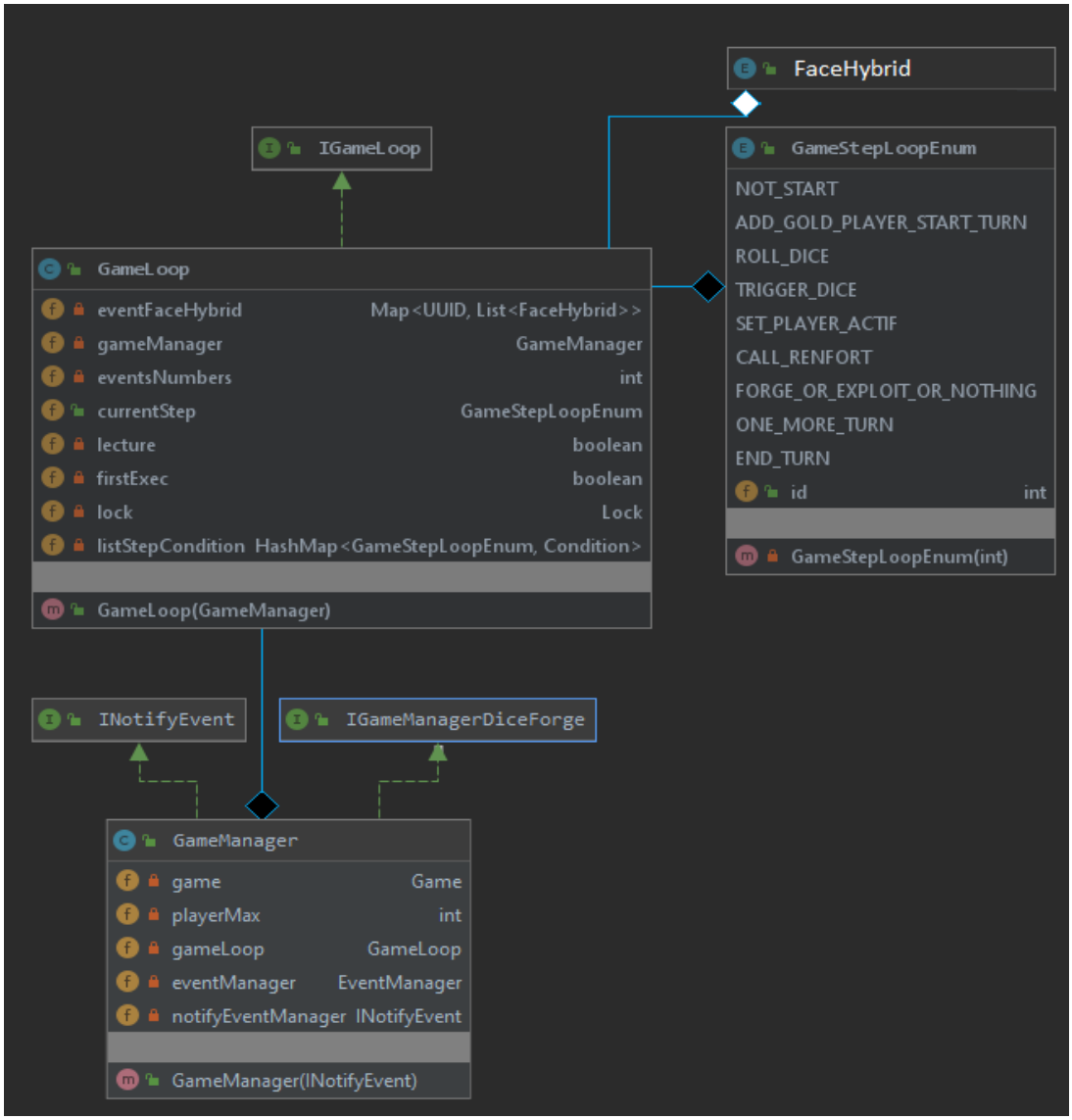
Point de vue statique

1. Les événements et actions



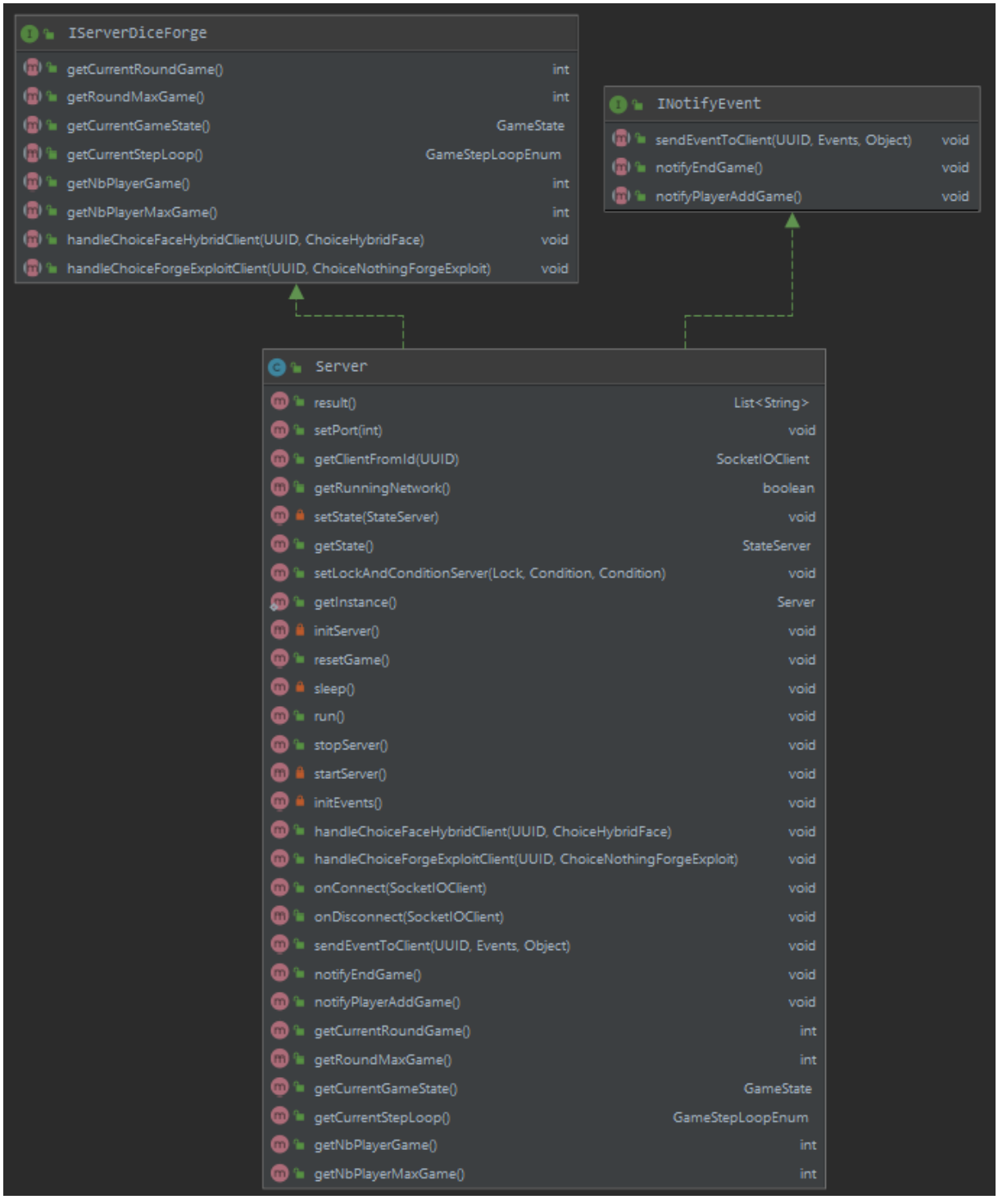
Le serveur utilise un design pattern observeur afin de gérer tous les événements liés au jeu. Chaque événement est lié à une action, qui sera invoqué. Nous avons crée une classe par Evenements et donc par Action. Par exemple nous avons une classe “EventRollDice” qui hérite de la classe “Event”, et une classe action complémentaire: “ActionRollDice” qui hérite de la classe Action, et qui sera invoquée par l'événement.

2. La boucle principale et le manager



La boucle de jeu principale aide le game manager a géré tous les traitements du jeu, ainsi qu'un "current state" qui correspond à l'état actuel du jeu (lancé de dé, utilisation de la forge, exploits …).

3. Le serveur



Le serveur s'occupe de gérer les événements (envoi des événements au clients, et handle des événements reçu), en fonction de l'état du jeu ("current state") et fais les redirections en fonction.

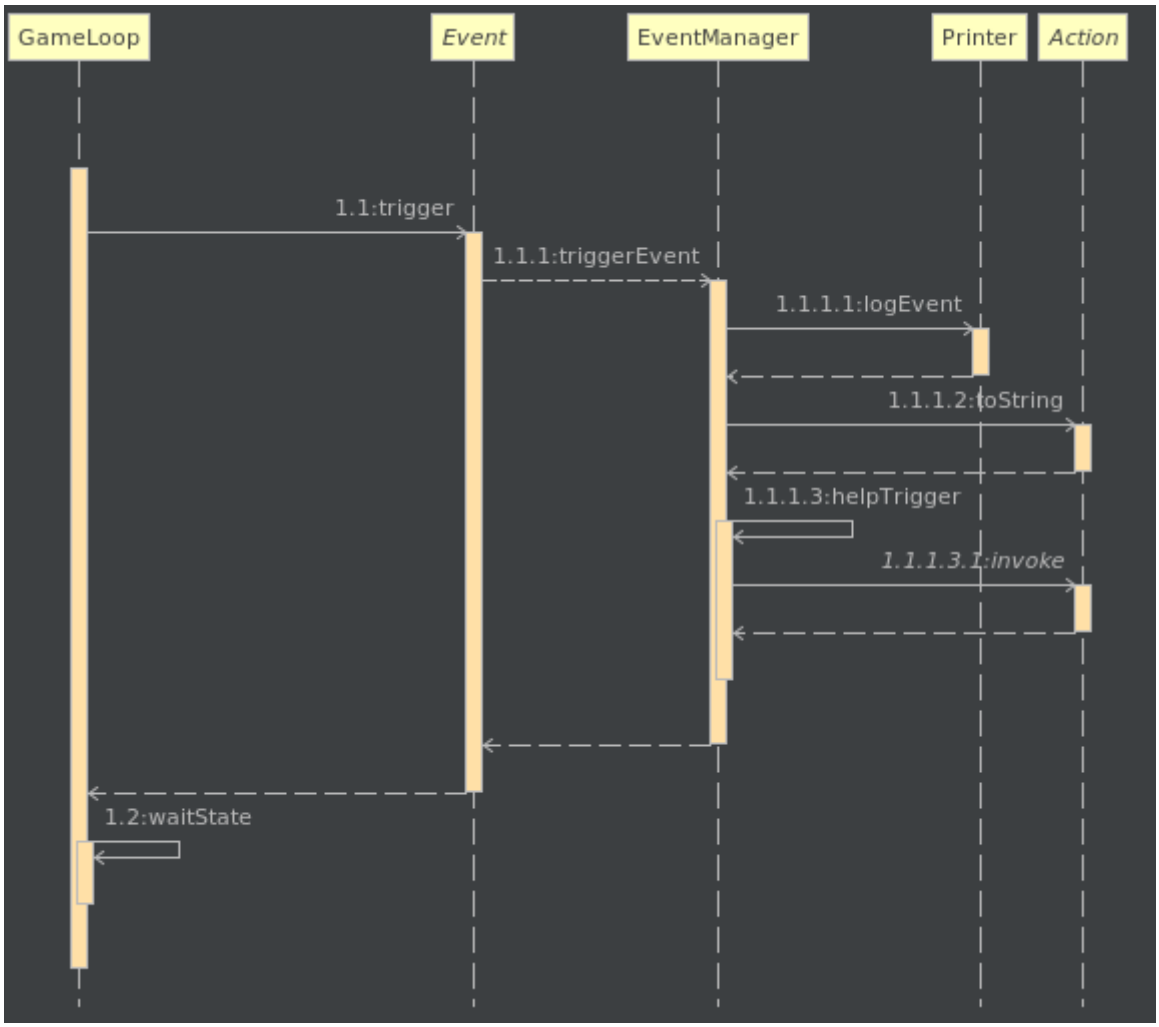
Point de vue dynamique

ChoiceForgeOrExploiOrNothing

Pré-condition :
Le client est connecté
Le client doit choisir une action

Post-condition :

Le serveur exécute l'action choisie



Le serveur invoke l'action choisie par le client, et gère son choix.

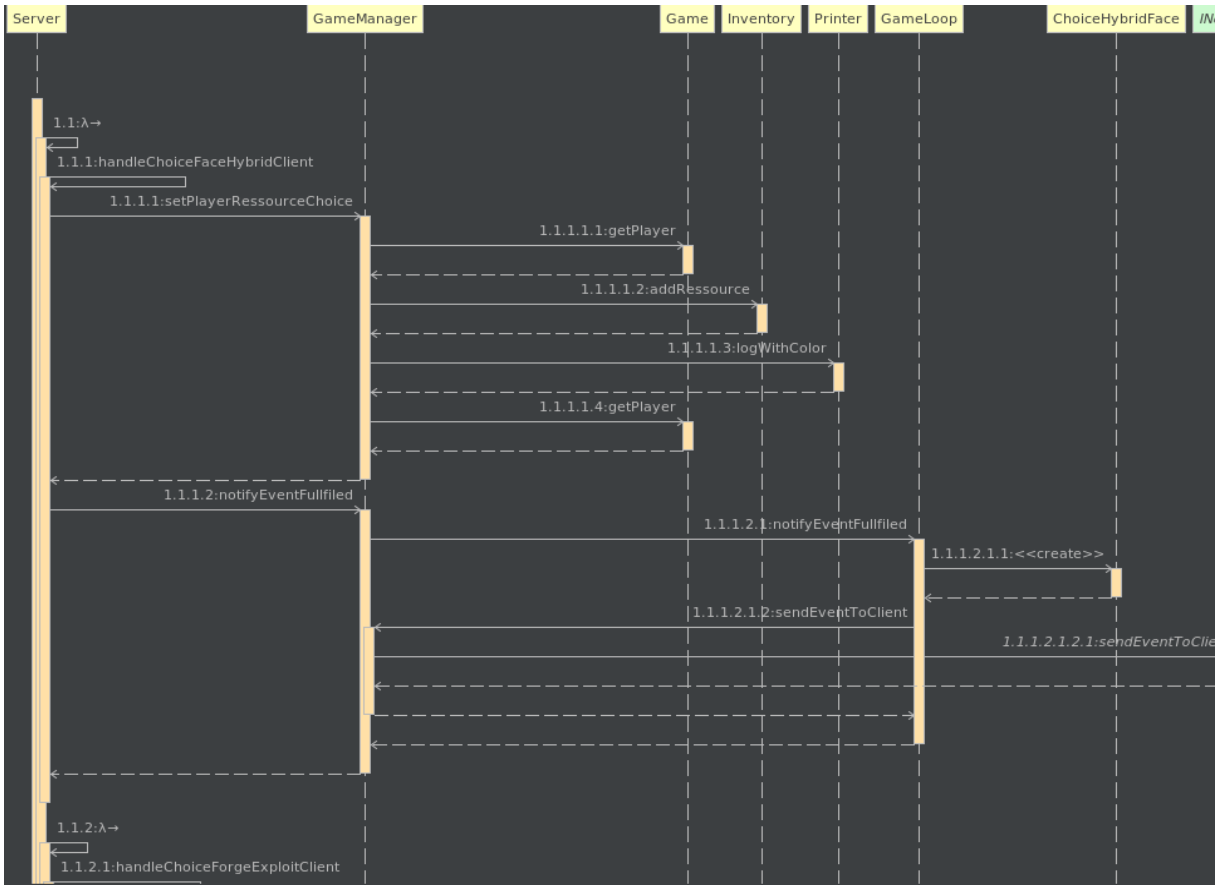
ChoiceRessourceFaceHybrid

Pré-condition :

Le client est connecté
Le client doit choisir une ressource dans sa face

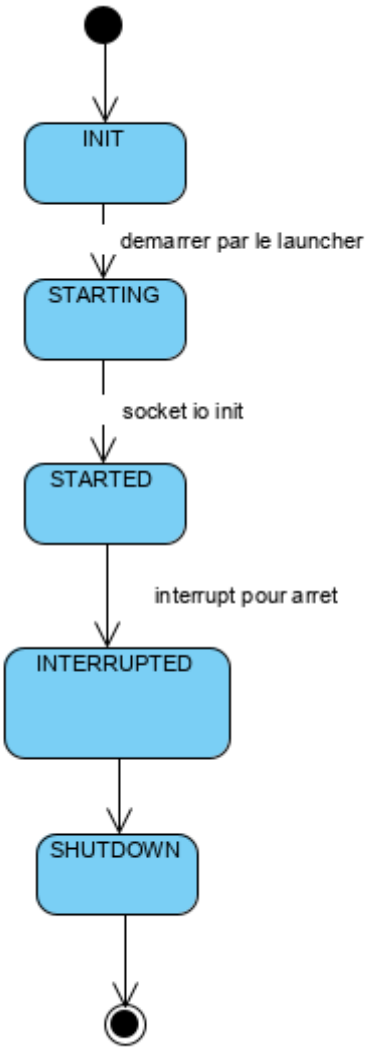
Post-condition :

Le serveur ajoute la ressource choisie à l'inventaire du client

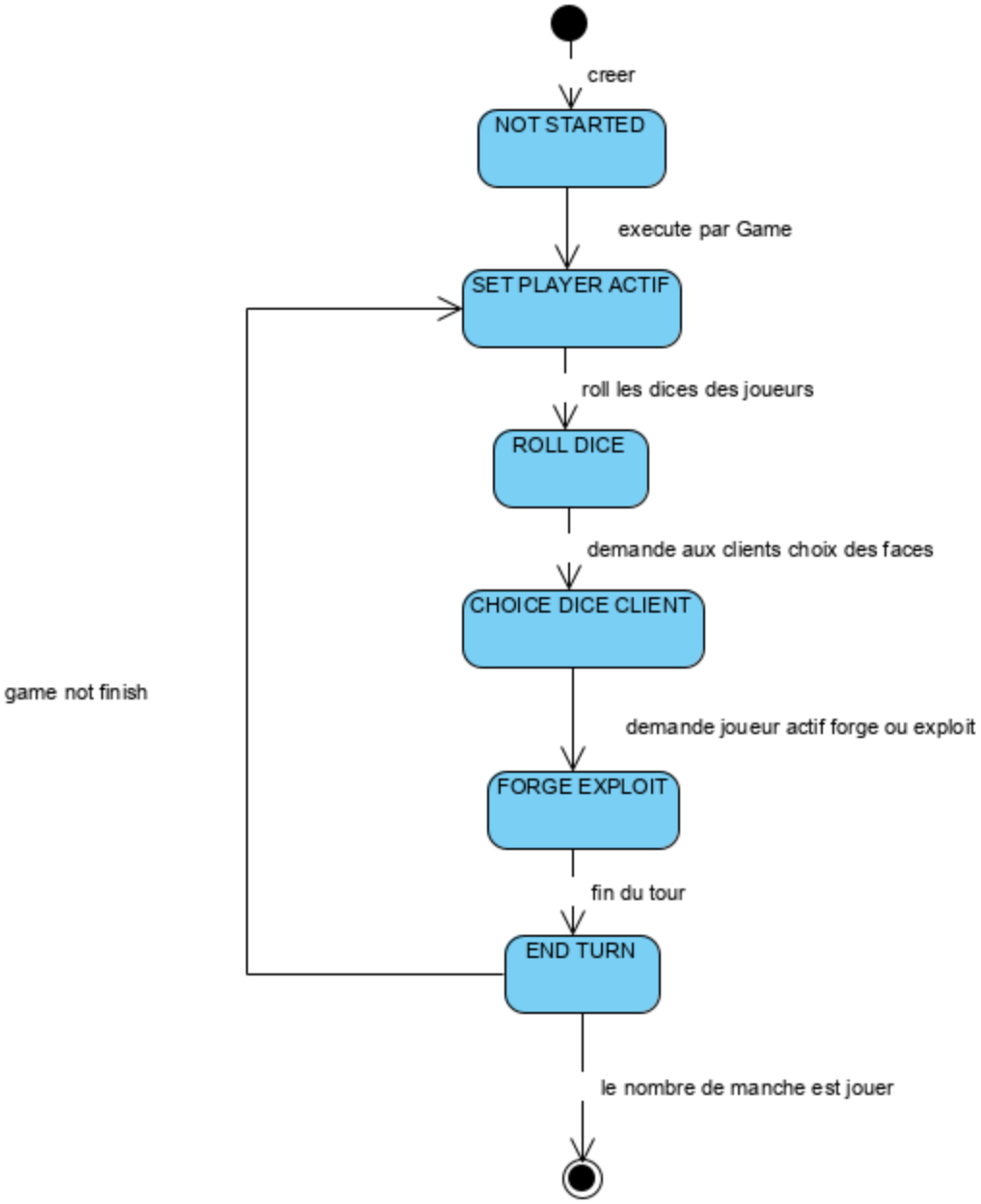


Le serveur gère le choix du client, modifie son inventaire, et affiche son choix dans la console puis notifie le client.

STATE DIAGRAM SERVER



STATE DIAGRAM GAME LOOP

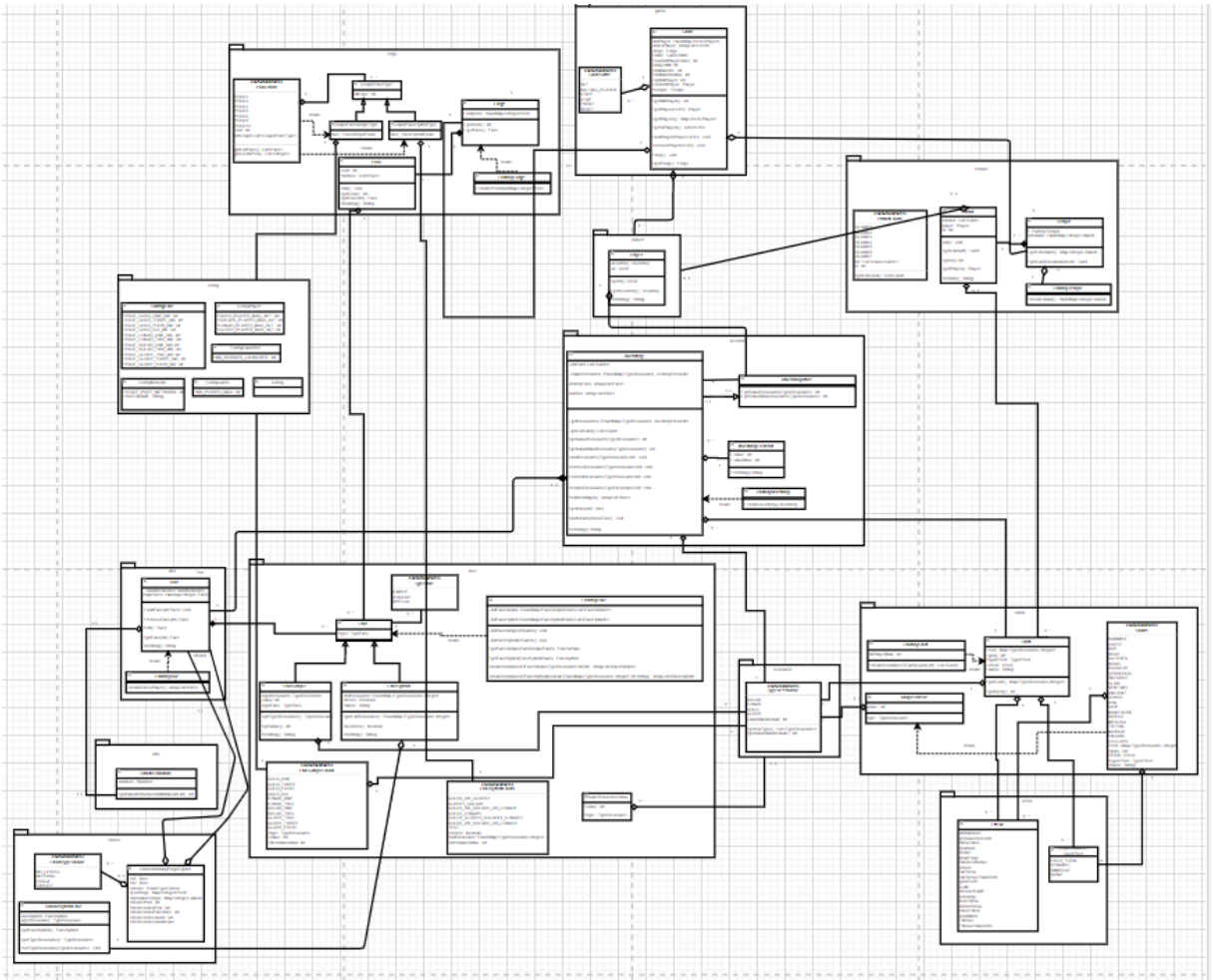


Module Share

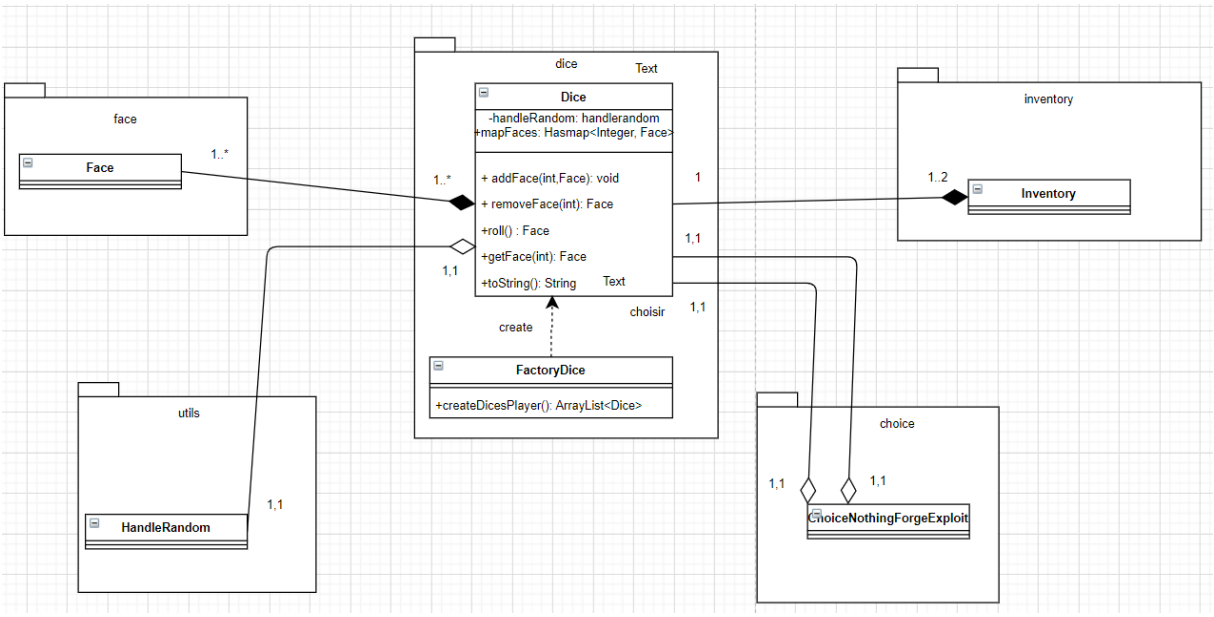
Conception logiciel

Point de vue statique

Représentation générale du package share :

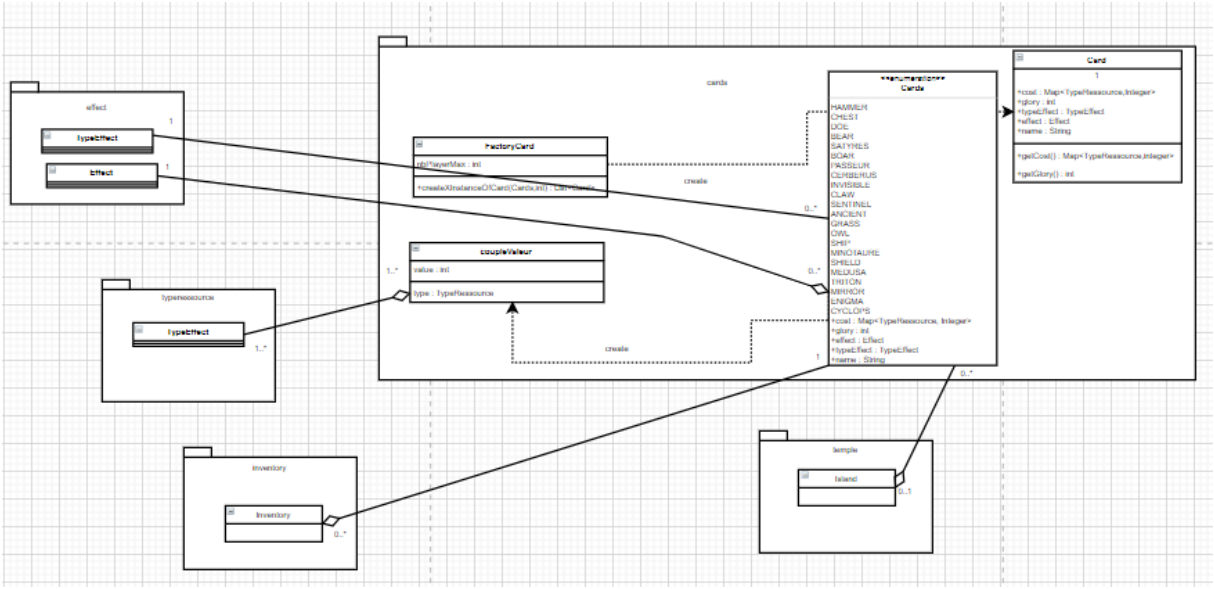


Gestion des dès



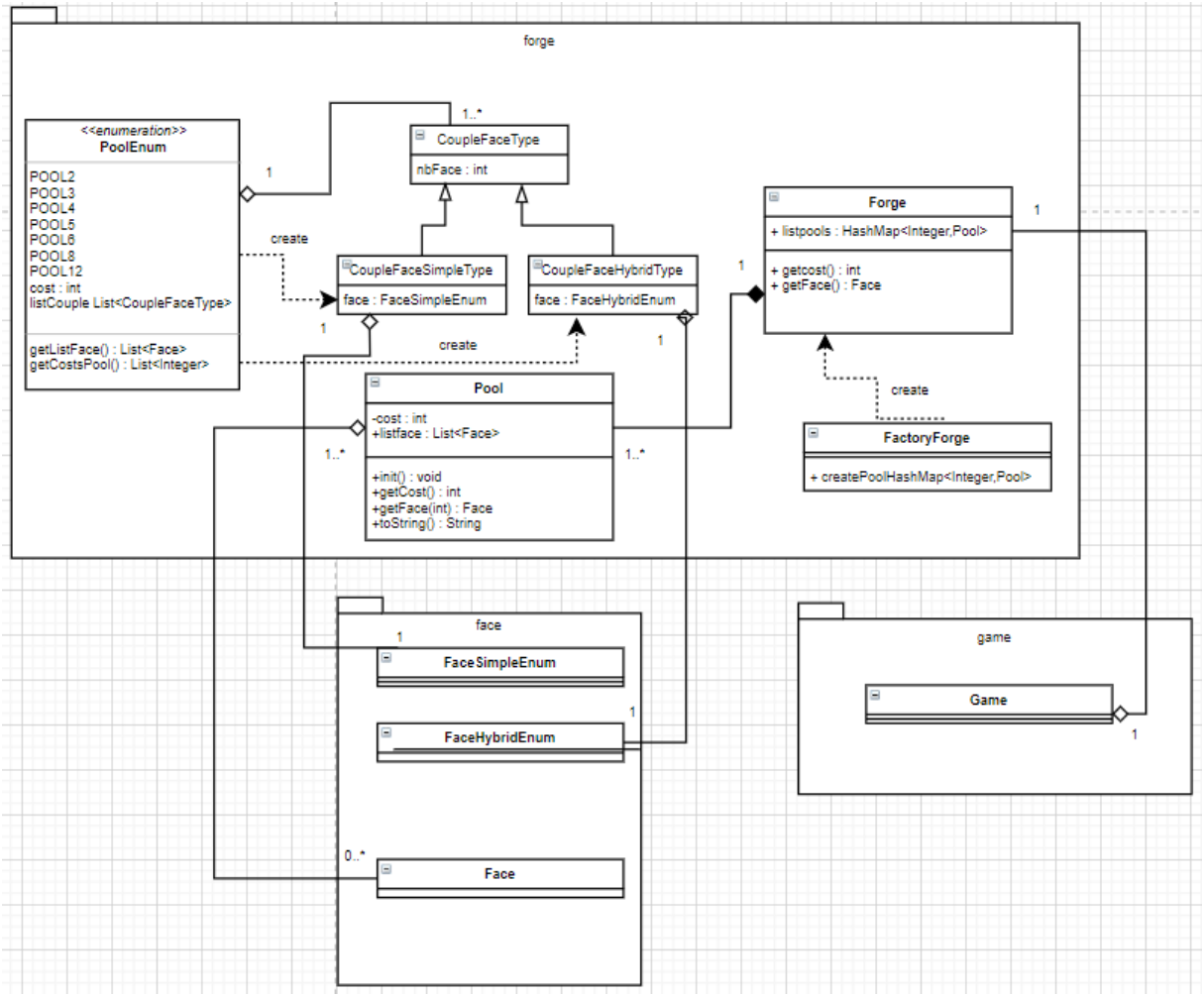
Un dé est obligatoirement rangé dans un inventaire, un inventaire peut accueillir 2 dés. Un dé est composé de plusieurs faces et une face peut composer plusieurs dés. Lorsqu'un dé est lancé (roll) il fait appel à la fonction HandleRandom. FactoryDice permet de créer les 2 dés d'un joueur.

Gestion des cartes



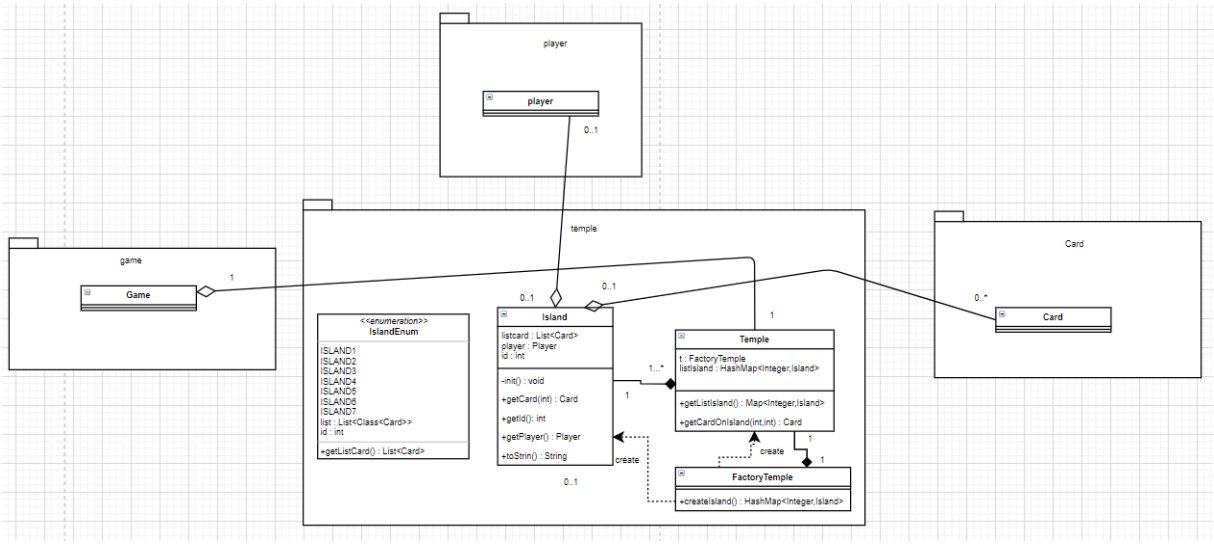
Une carte peut se trouver dans une island ou dans un inventaire. Chaque carte a un effet et un type d'effet (recurrent, dynamique) une carte à un coût, l'énumération Cards permet de répertorier toutes les cartes d'une partie. FactoryCard permet de créer les cartes dans les islands au début de la partie.

Gestion de la forge



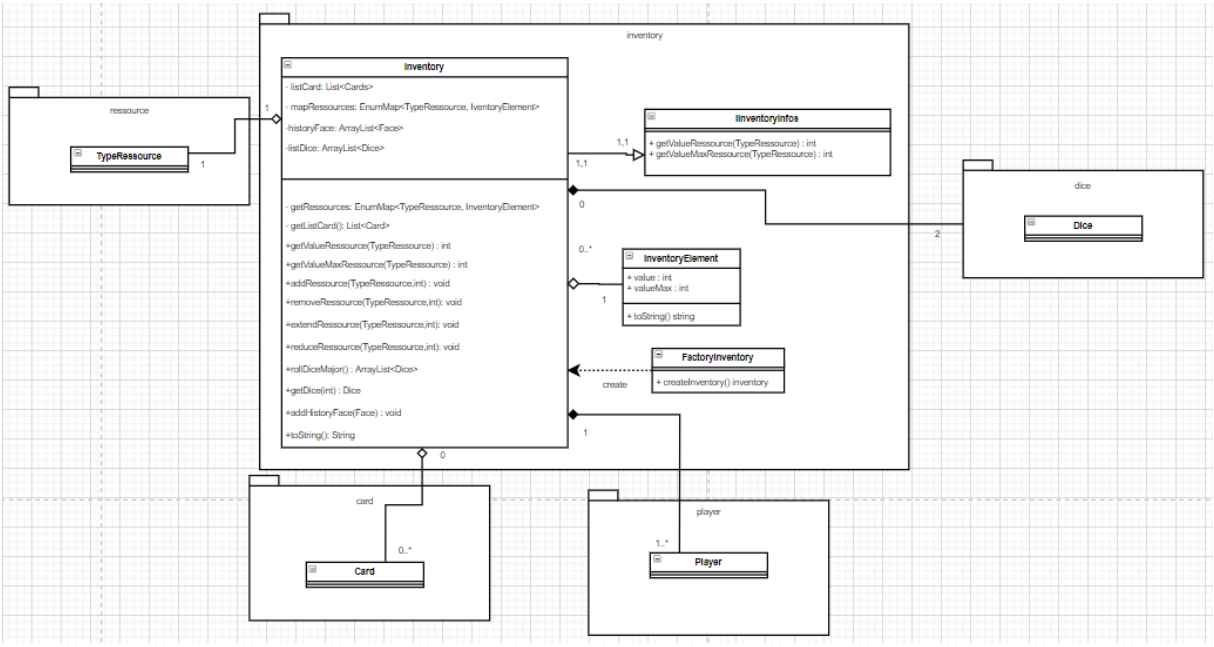
La forge est composée de 7 pools, chaque pool possède un coût et est constituée de différentes faces simple ou Hybrid qui sont dans une liste. Une forge est dans une seule partie et une partie est composée d'une seule forge. FactoryForge permet de créer la forge de la partie.

Gestion du temple



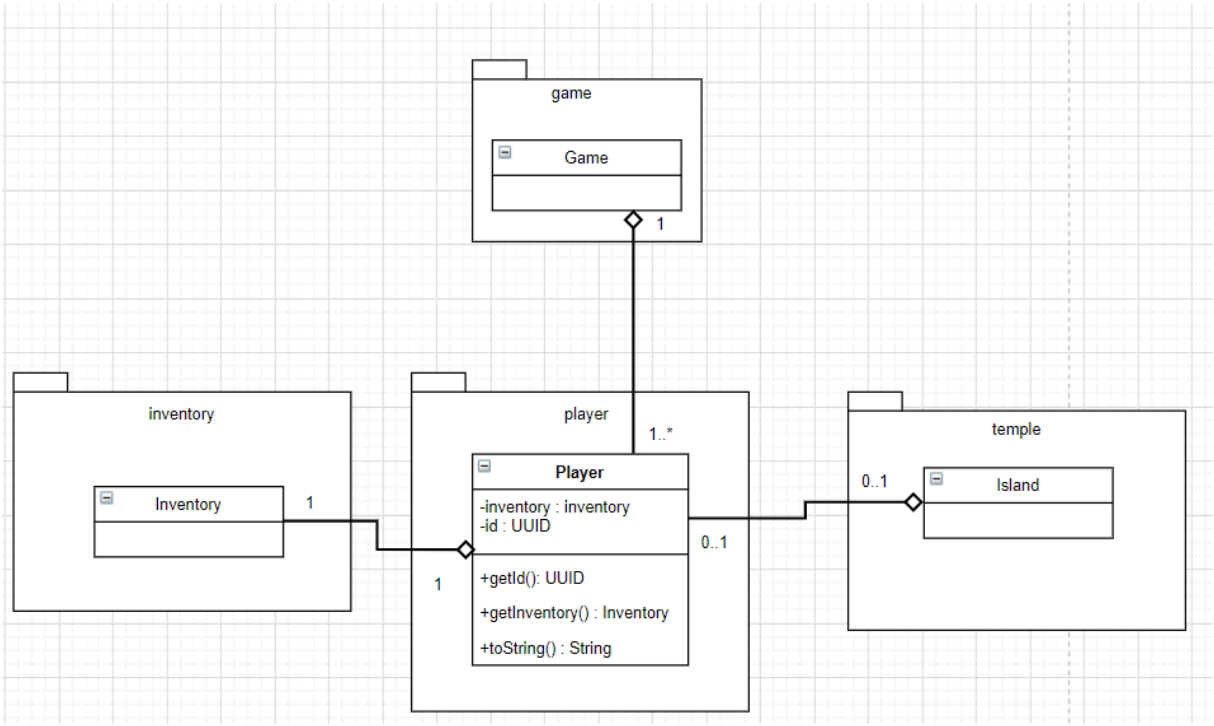
Un temple fait partie d'une seule partie et une partie possède un seul temple. Un temple est composé de plusieurs islands qui peuvent accueillir un seul joueur et sont composées de plusieurs exemplaires d'une carte. FactoryTemple permet de créer le temple de la partie.

Gestion de l'inventaire



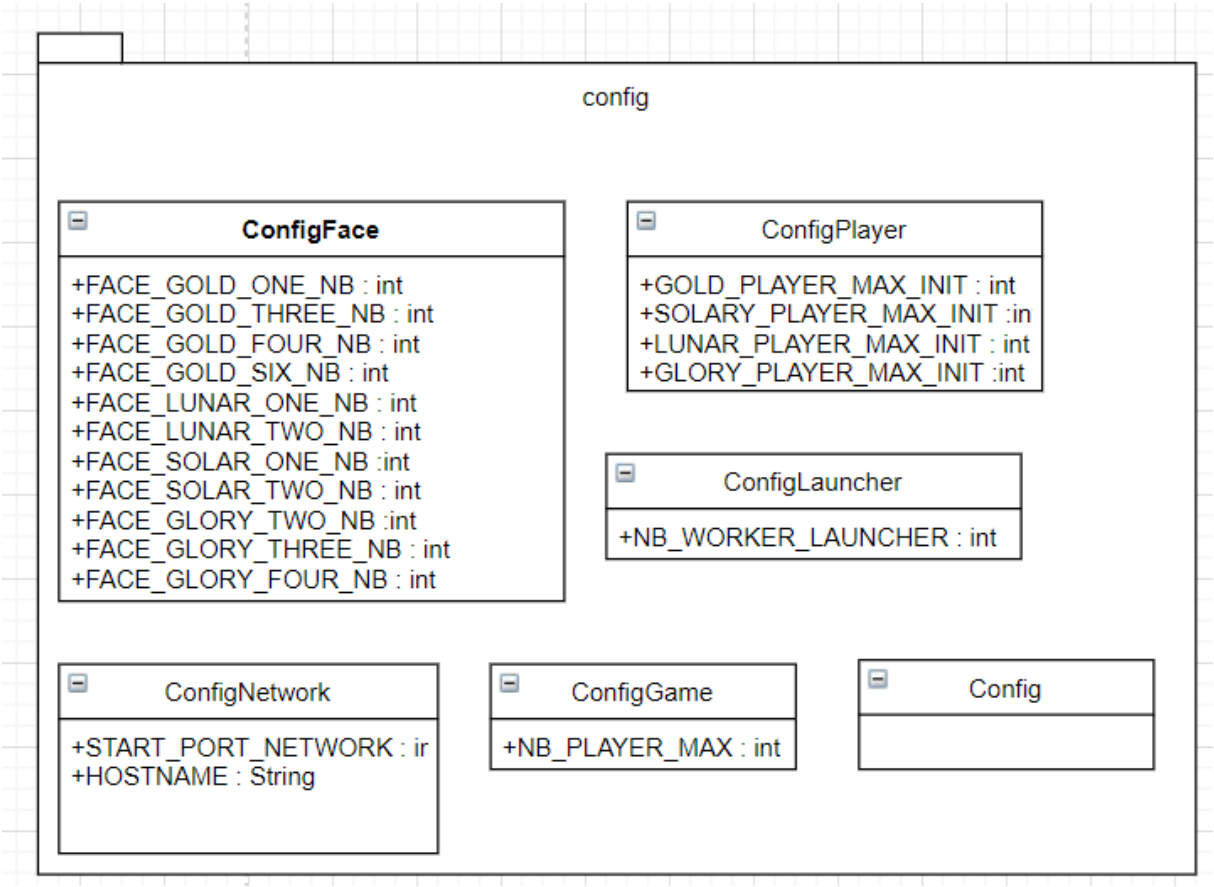
Un inventaire est obligatoirement relié à un joueur, un inventaire possède plusieurs types de ressources qui ont une valeur courante et une valeur maximale, l'inventaire est aussi composé de 2 dés, il peut contenir les cartes des exploits accomplies par le joueur. FactoryInventory permet de créer l'inventaire d'un joueur.

Gestion du joueur



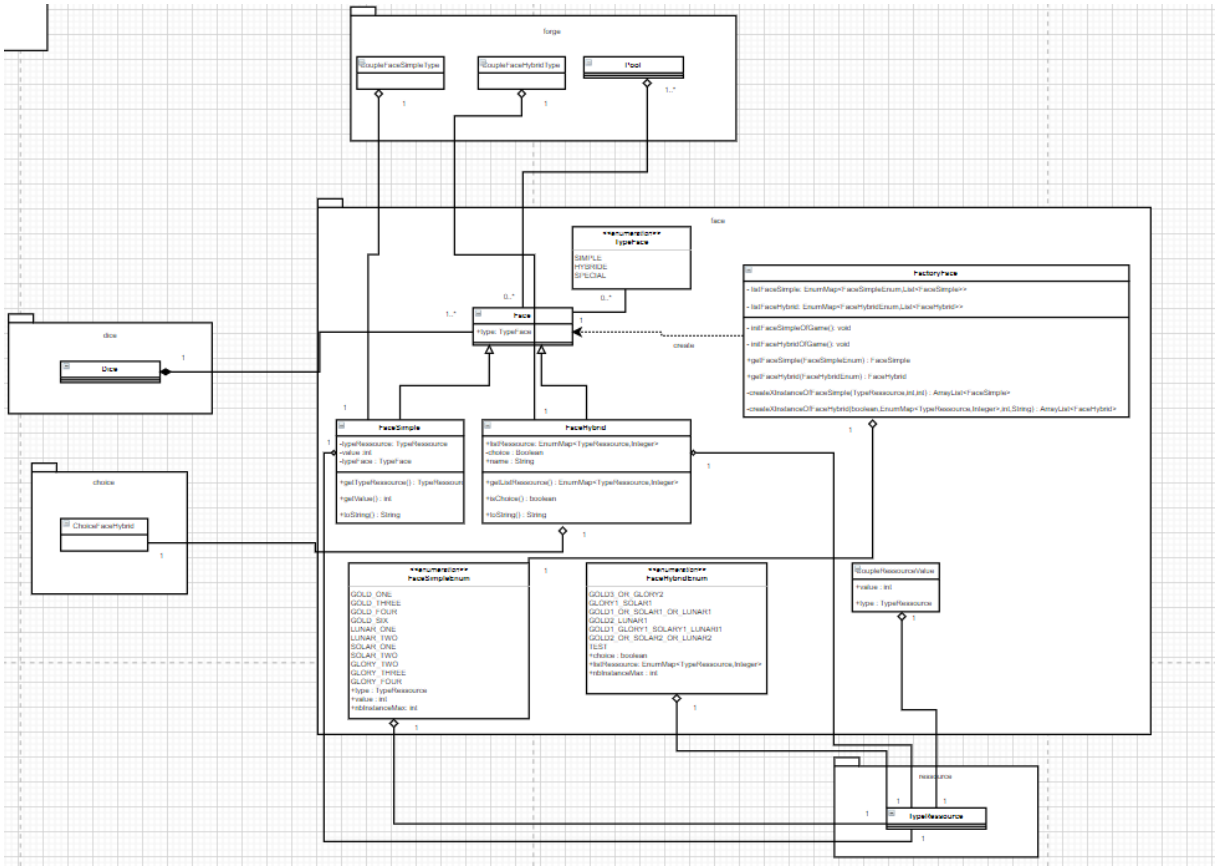
Un joueur possède un seul inventaire, s'il a effectué un exploit il sera placé sur une île. Un joueur est connecté a une seule partie et une partie peut accueillir plusieurs joueurs.

Config



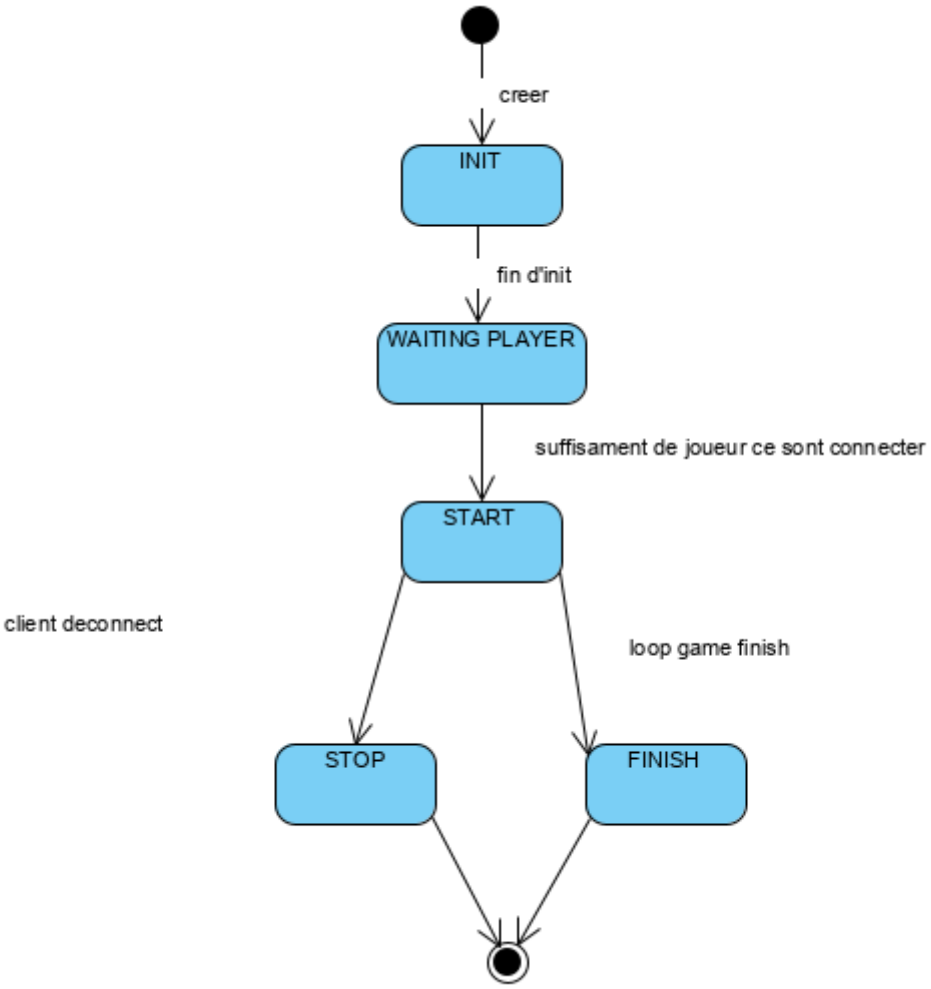
La config contient les différentes faces possibles, le nombre de joueurs max d'une partie, le nombre maximale des réserves de gold,lunar et solar et de glorypoint (point de victoire). La config contient aussi le port du réseau et le nom de l'hôte du réseau.

Gestion des faces



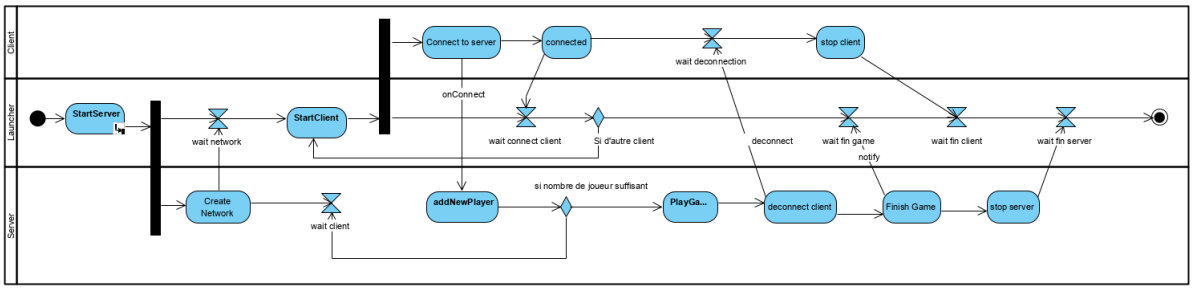
Toutes les faces sont composées d'un typeface d'un couplevaleur ou d'une liste de couplevaleur. Les faces hybrides oblige à effectuer un choix entre plusieurs typeressources. Une face est soit sur un seul dé soit dans une pool de la forge. FactoryFace permet de créer les faces d'un dé ou d'une pool.

State diagramme game



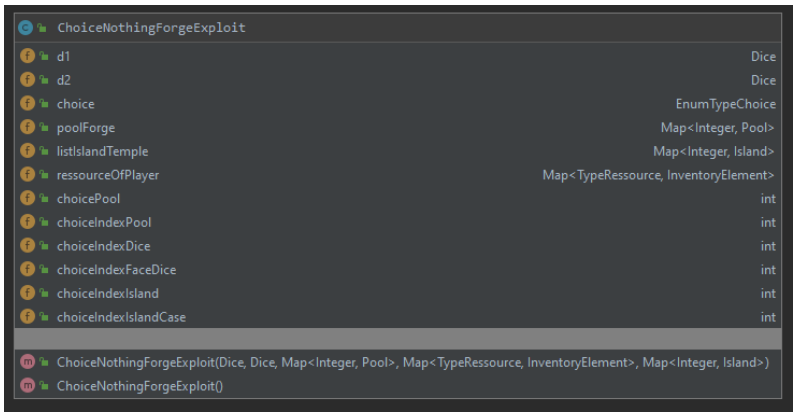
Interaction client serveur

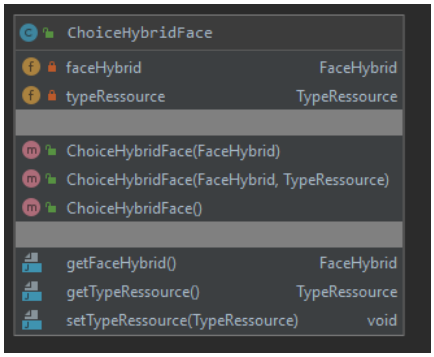
Interaction entre client, server, et launcher

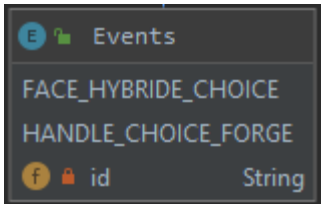


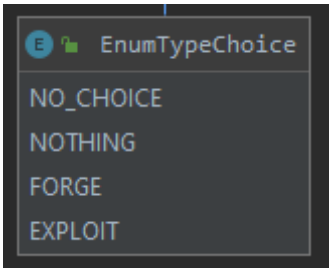
Ce diagramme re présente les interactions entre les différents acteurs du système de la connexion a la fin de partie de façon global.

Objet reseau / protocol



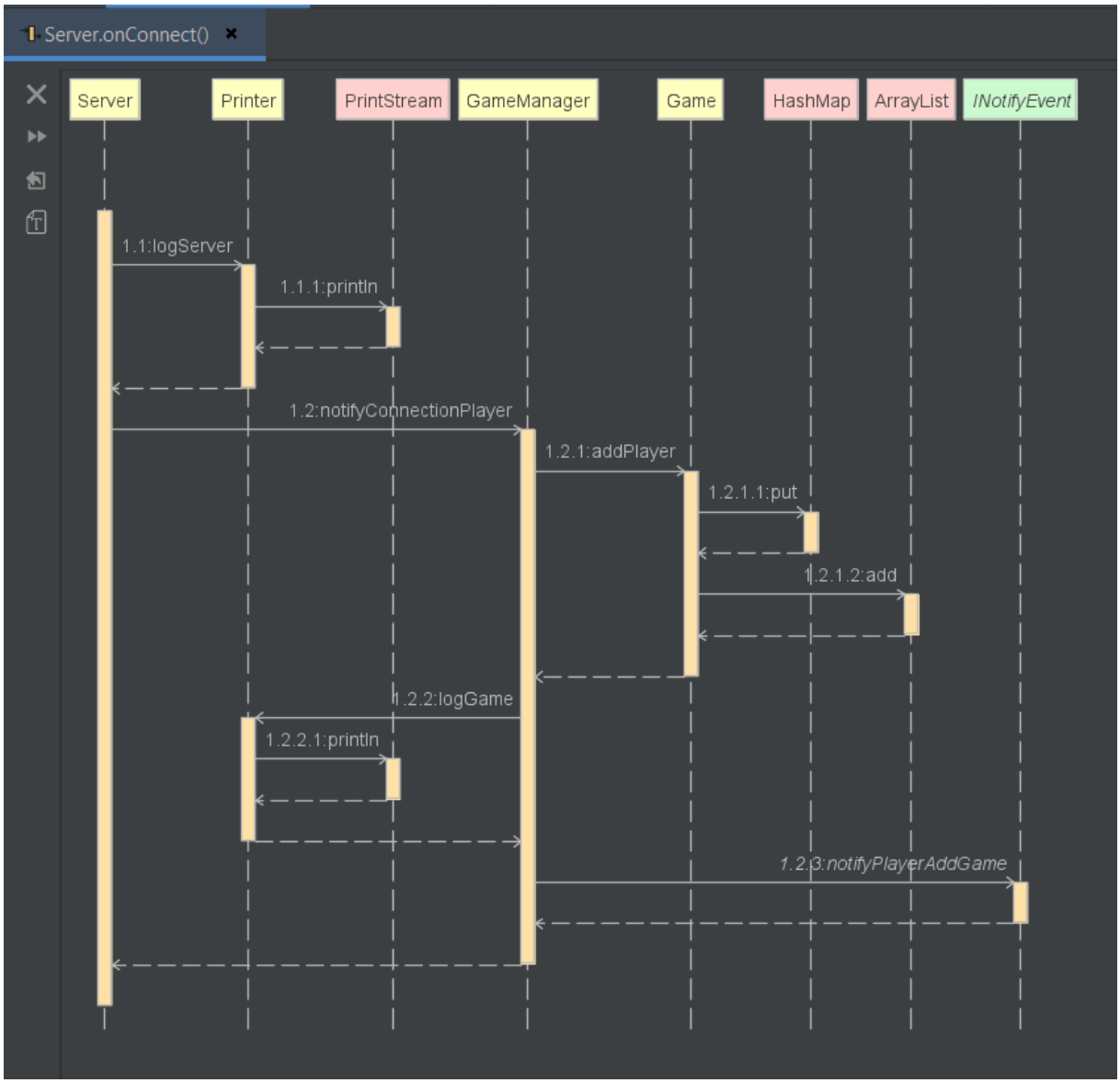




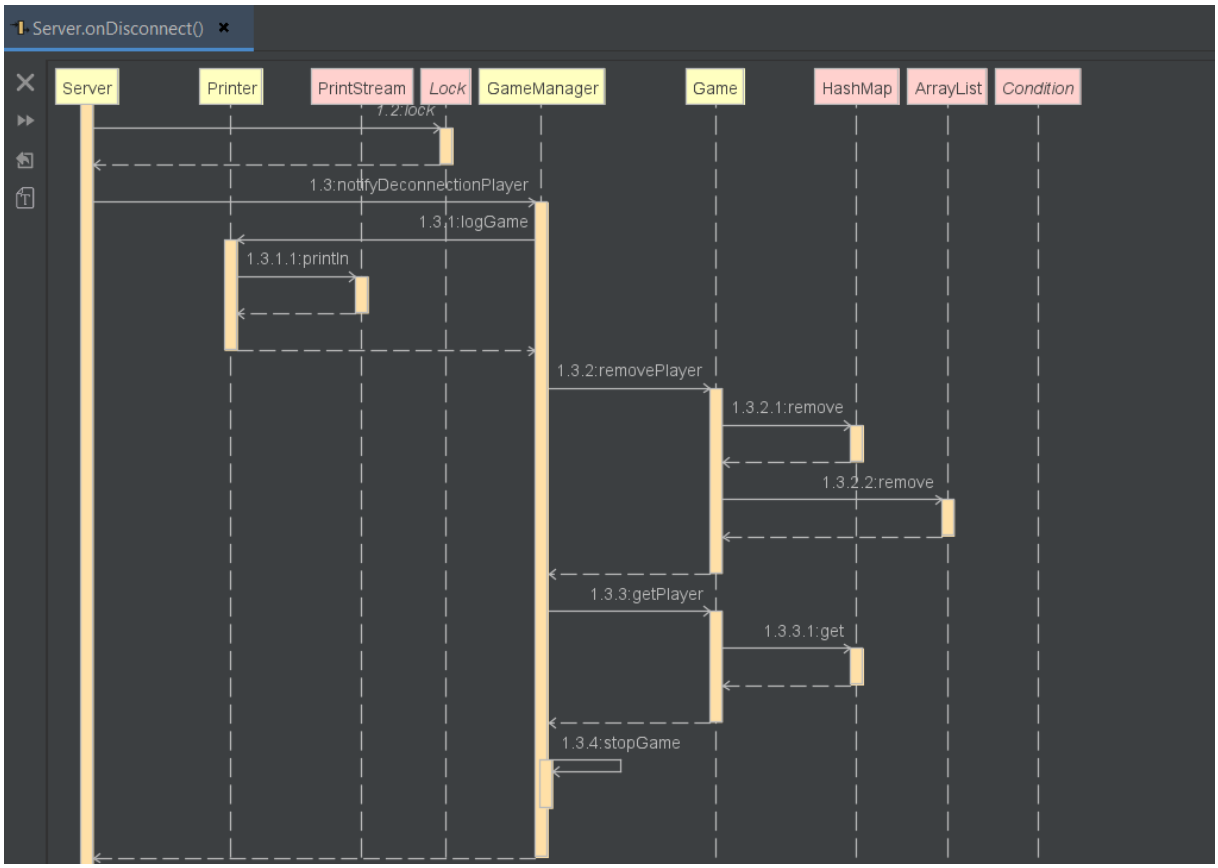


Enumération Events permet de renseigner tous les events réseaux entre le client et le serveur.

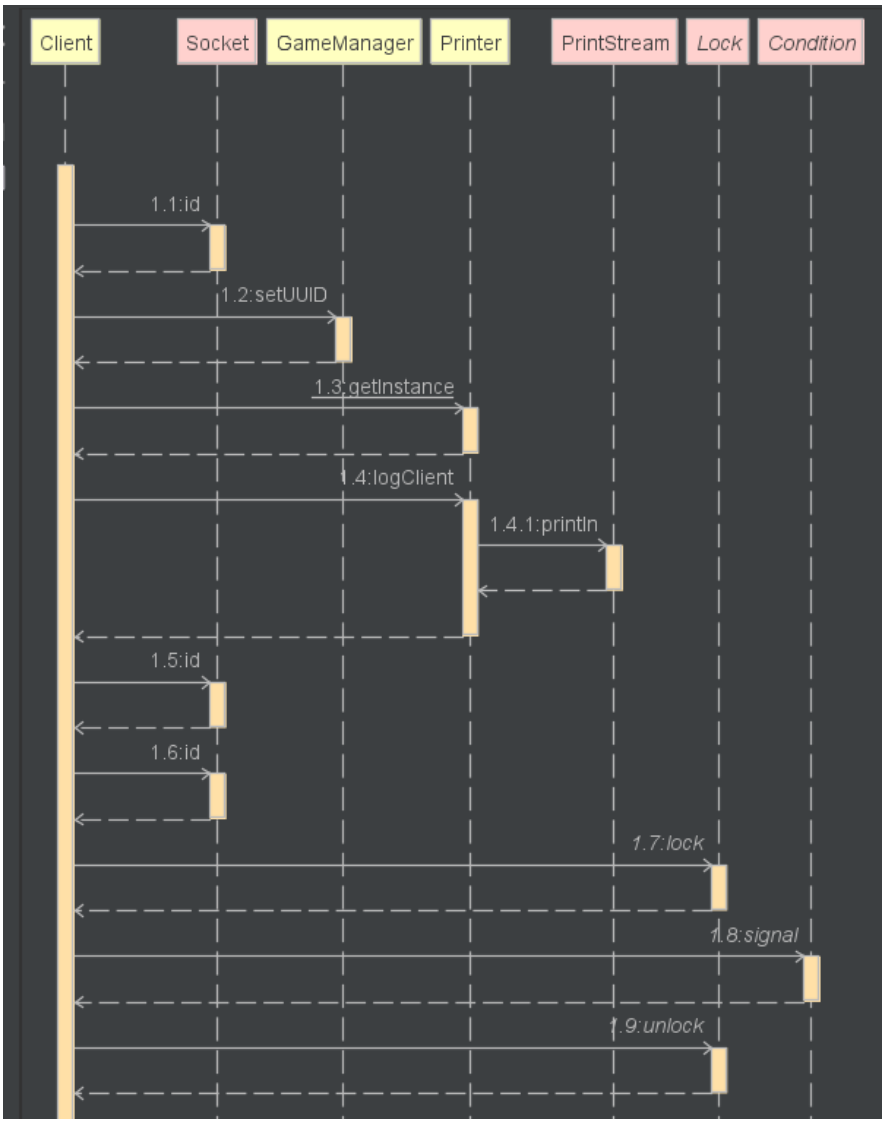
On connect serveur



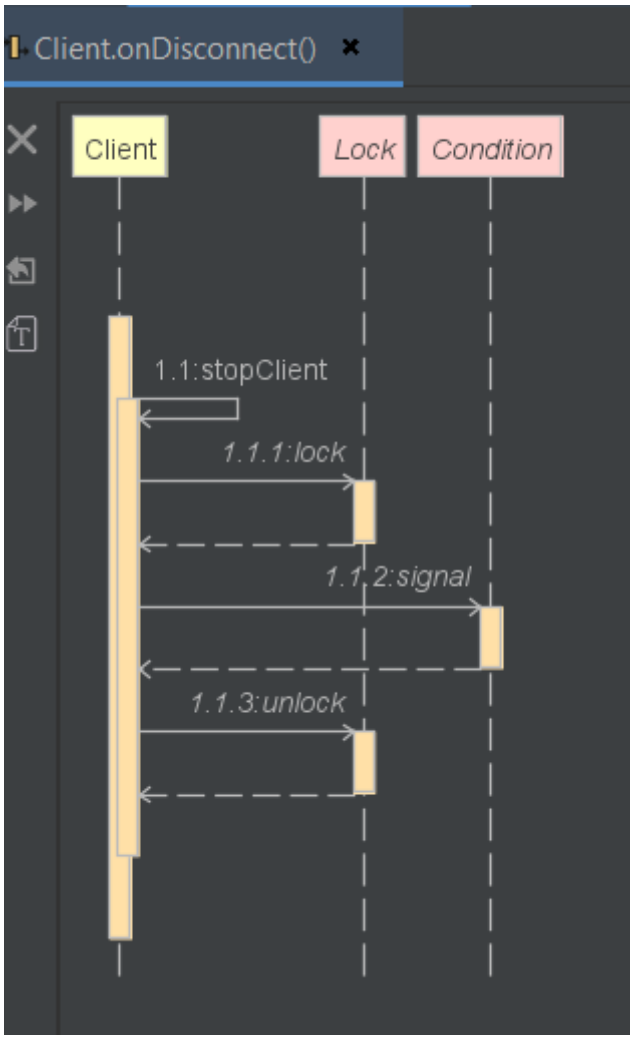
On disconnect Server



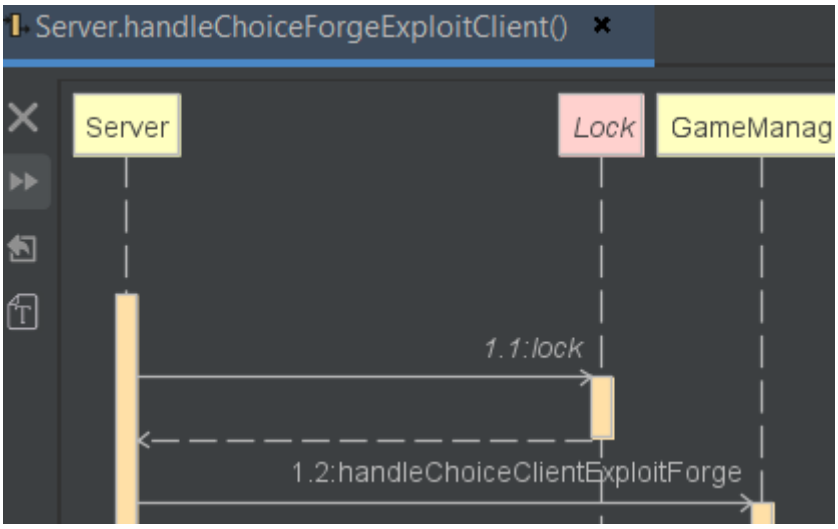
On connect client

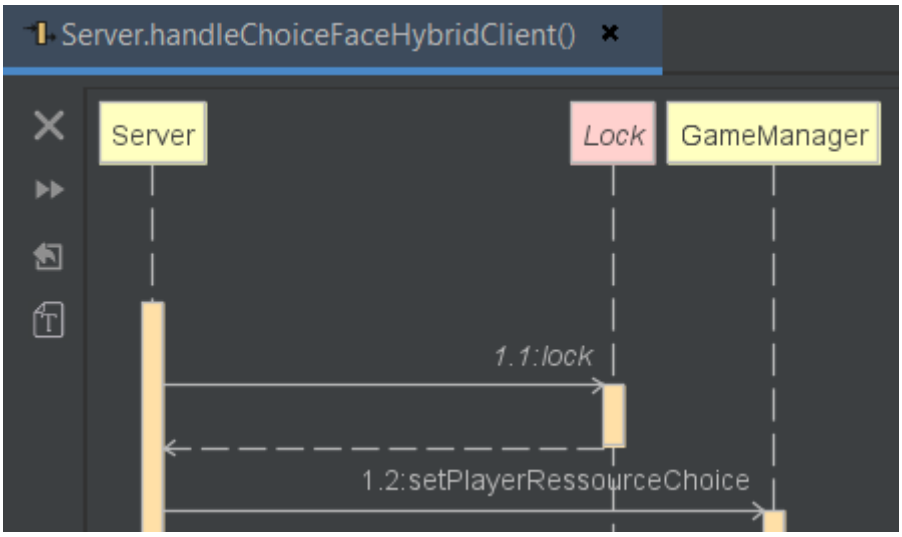


On disconnect client

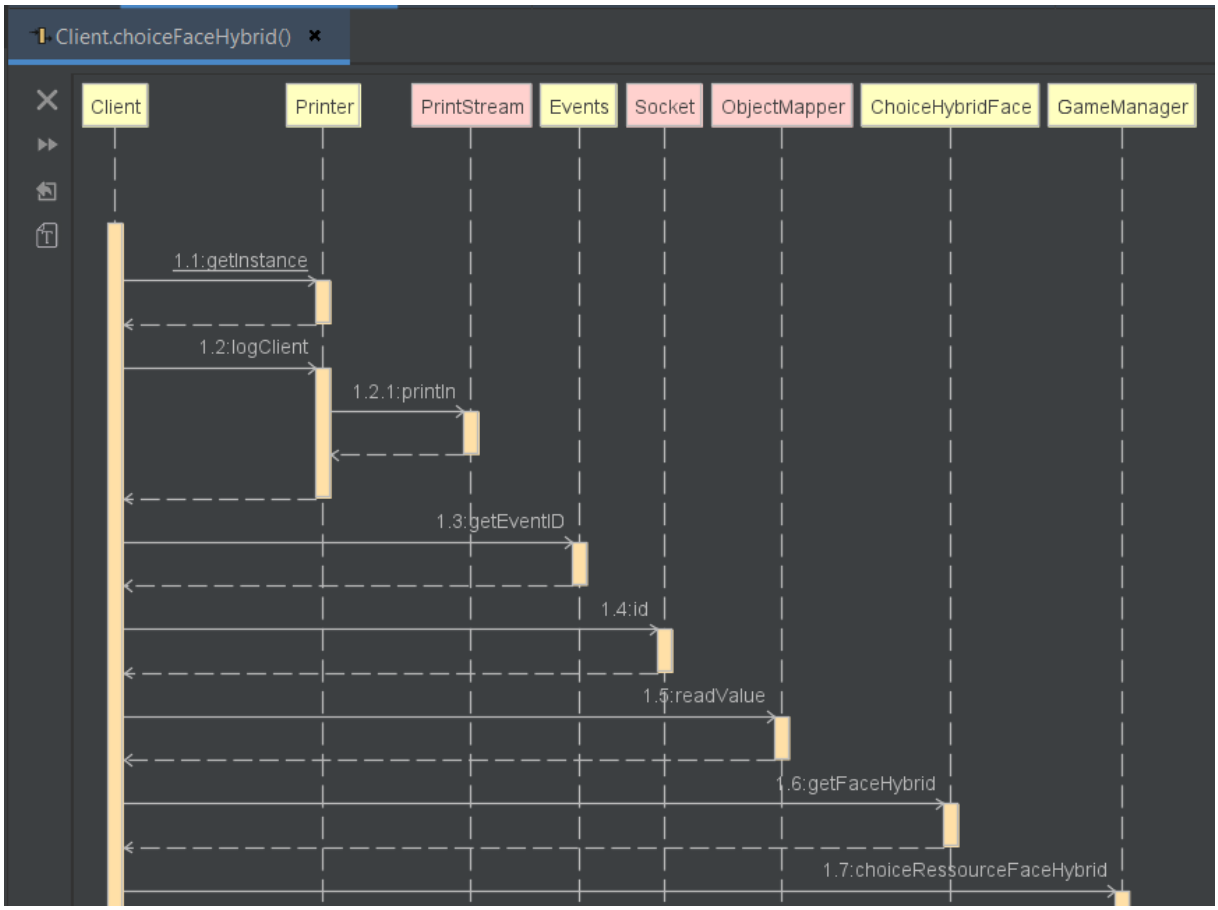


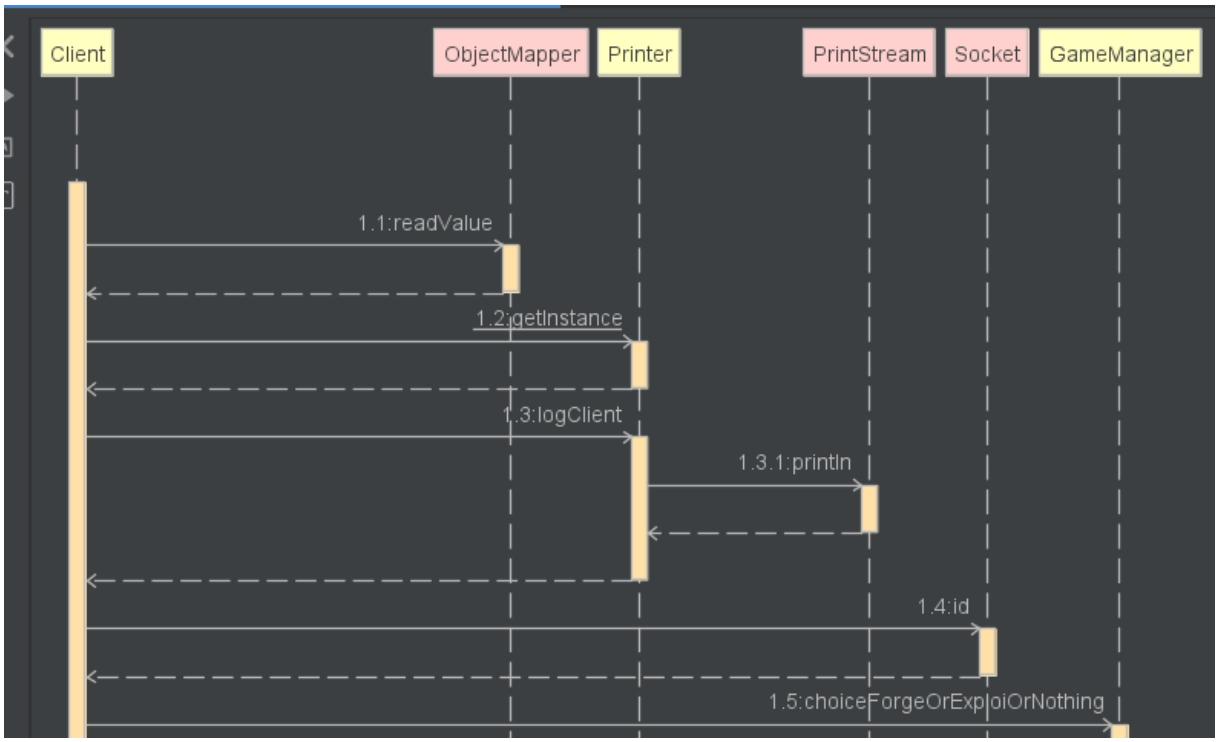
Handle Event Server





Handle event Client





Conclusion

5. Conclusion

Analyse de votre solution : points forts et points faibles

Nos points forts d'après nous est l'architecture assez modulaire et flexible, ainsi que le nombre de fonctionnalités, réseau client server et multi partie.

Notre point faible est justement a cause d'un de nos point fort étant le nombre de fonctionnalité a concevoir au niveau de la COO qui requiert beaucoup trop de travail à réaliser dans les temps imparties. Et peut être aussi la complexité qui monte au fur et à mesure.

Évolution prévue

Finir l'implémentation total des cartes ainsi que leurs effets et aussi les statistiques.