Diceforge201920

2ème rendu intermédiaire

Groupe : TP2 DFA

Équipe

GILLES Marco

LEFEUVRE Maxence

MERCURI Sabrina

CAMORANI Alexis

DESIRE Stéphane

Professeur référent

FERRY Nicolas

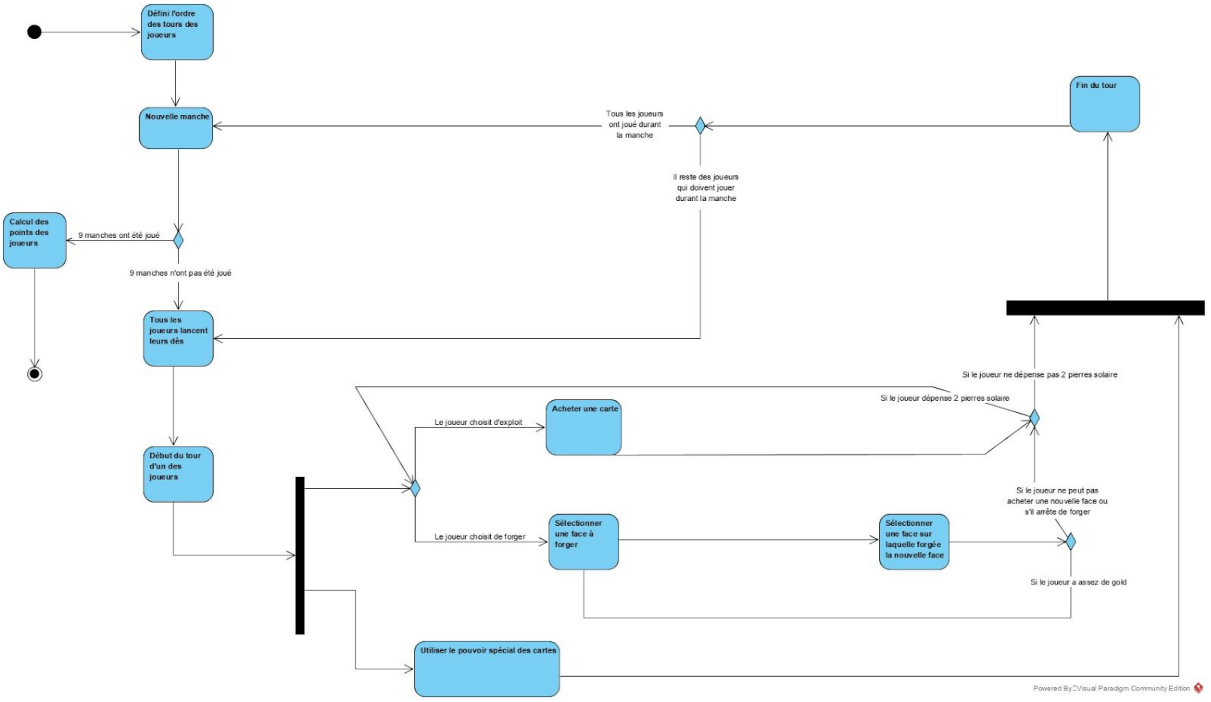
04/03/2020

Point de vue général

Glossaire

Joueur : Quand on parlera de Joueur on parlera de la classe java Joueur pas de l'utilisateur
Utilisateur : L'utilisateur est la personne/l'entité qui jouera avec le Client
Client : Le Client désigne le programme qui sert à l'utilisateur pour pouvoir communiquer avec le serveur.
Forge : La forge ou aussi le sanctuaire.
Sanctuaire : L'endroit où le joueur peut acheter des faces de dé
Catégorie de la forge : Les différents pallier de coût pour acheter une face

Représentation générale



Fonctionnalités traitées

Fonctionnalités traitées
Durant les itérations 1, 2, 3 nous avons traité les fonctionnalités suivantes :

- Connexion du client au serveur
- Le serveur peut accepter ou refuser les connexions des clients
- Attribution du pseudo de leur choix aux utilisateurs
- Démarrage de la partie
- Modélisation de la forge
- Modélisation des différentes catégories de la forge
- Modélisation des îles
- Création des cartes (sans les effets pour le moment)
- Création de l'inventaire
- Création des dés et des faces
- Lancer des dé et transmettre les résultats des lancers

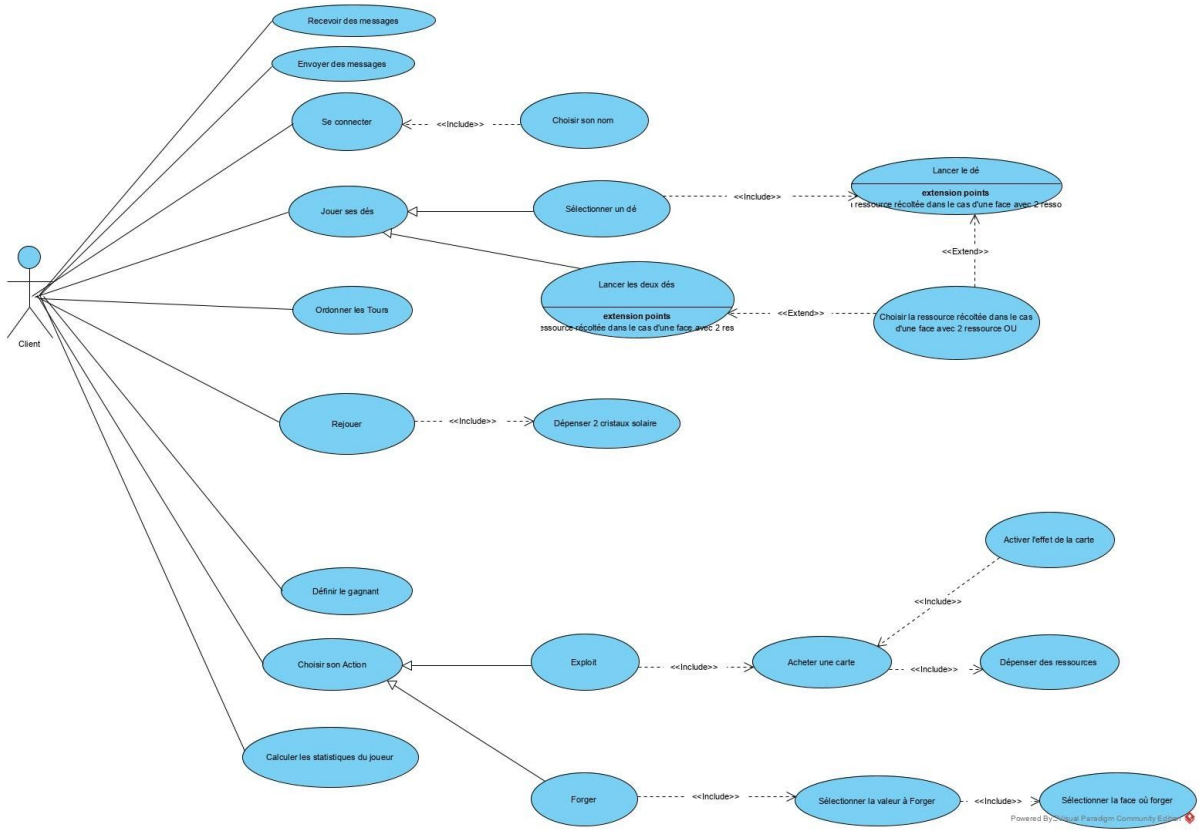
- Acheter des faces de dé et transmettre l'achat de la face
- Placer les faces de dé et transmettre le placement de la face
- Acheter des cartes et transmettre l'achat de la carte
- Condition de victoire (pour le moment sur un nombre X de points de victoire)

Durant les itérations 4, 5 nous avons traité les fonctionnalités suivantes :

- Condition de victoire en fonction du nombre de points de victoire à la fin de 9 manches
- Synchronisation du client et du serveur quant à l'éxécution des tours et des manches
- Ajout de statistiques
- Séparation du projet en plusieurs modules avec chacun leurs pom.xml
- L'ajout d'un plateau dans le serveur
- Création d'une CardEffectFactoryServer pour activer les effets des cartes
- Création des effets des cartes dans la CardEffectFactoryServer
- Création de nouveaux messages de communications entre le client et le serveur pour les effets des cartes

Use Case

Use case : Système Serveur



Par rapport au rendu précédent, nous avons revu nos Use Case. En effet pour le système Serveur, l'acteur doit être le Client et nous devons réfléchir de la façon suivante : "Quels services rend le système à l'acteur". De ce fait nous en avons conclu que le serveur rend comme service au Client le

fait de se Connecter, Recevoir/Envoyer des messages, Jouer ses dés donc de Lancer 1 ou 2 dés, de Rejouer, jouer une carte, Choisir son action, Acheter une carte et jouer son effet, Forger des faces, etc . Comme nous avons ajouter à notre projet les effets des cartes et leur activation, nous avons donc aussi ajouter au Use Case le fait d'Activer l'effet de la carte. de plus le serveur ordonne aussi les tours des joueurs et calcul les statistiques des joueurs en plus de définir le gagnant.

User story

Titre : Lancer un dés

Description :

En tant que joueur,

je souhaite pouvoir lancer les deux dés de mon inventaire

afin de pouvoir obtenir des ressources.

Tests d'acceptances

Scénario :

Etant donné que j'ai lancé les dés

Et que j'ai obtenu une ressource sur chaque dé

Quand je veux récupérer les ressources

Alors je les récupère

Scénario :

Etant donné que j'ai lancé les dés

Et que j'ai obtenu une ressource sur un dé et un multiplicateur sur l'autre

Quand je veux récupérer la ressource

Alors je la récupère autant de fois que le multiplicateur

Scénario :

Etant donné que j'ai lancé les dés

Et que j'ai obtenu un multiplicateur sur les 2 dés

Alors rien ne passe

Scénario :

Etant donné que j'ai lancé les dés

Et que j'ai obtenu une ressource ou un multiplicateur sur un dé et que j'ai obtenu un portail sur l'autre

Quand je veux récupérer les ressources

Alors je sélectionne une ressource chez un autre joueurs et je la récupère en appliquant le multiplicateur si j'en ai obtenu un, sinon je la récupère simplement.

Titre : Acheter des faces de dés

Description :

En tant que joueur je peux changer la face d'un de mes dés afin de l'améliorer.

Test d'acceptances :

Scénario :

Etant donné que je veuille forger un dé, si j'ai assez de ressources pour

avoir l'amélioration choisie

Alors je sélectionne la face de mon dé que je veux changer, puis la nouvelle face de mon dé.

La nouvelle face prend la place de l'ancienne.

Titre : Acheter des cartes

Description :

En tant que joueur,

je peux acheter des cartes afin de pouvoir

améliorer mon équipement ou gagner des points de victoire.

Test d'acceptances

Scénario :

Etant donné que j'ai réuni assez de ressources

je peux acheter une carte.

Je l'obtiens.

Titre : Rejouer

Description :
En tant que joueur,
Je souhaite dépenser 2 solary stone à la fin de ma première action
Afin de pouvoir refaire une action.

Titre : Calculer les statistiques

Description :
En tant que statisticien,
Je souhaite obtenir les statistiques des résultats de plusieurs parties lancées d'affilés,
Afin de pouvoir déterminer qui est le meilleur joueur.

Titre : Ordonner la partie

Description :
En tant que joueur,
Je souhaite que ma partie se déroule dans l'ordre logique
Afin de pouvoir attendre les réponses des joueurs quant à leurs actions et que les joueurs puissent recevoir les ressources obtenus

Titre : Activer les effets

Description :
En tant que joueur
Je souhaite pouvoir activer les effets de mes cartes
Afin d'obtenir un avantage en jeu.

Critère d'acceptance :

- Les cartes ont les bons effets associer
- Le joueur peut activer les effets des cartes si c'est l'effet de la carte est activable à chaque tour
- L'effet de la carte s'active lors de son achat, si son effet s'active uniquement à l'achat

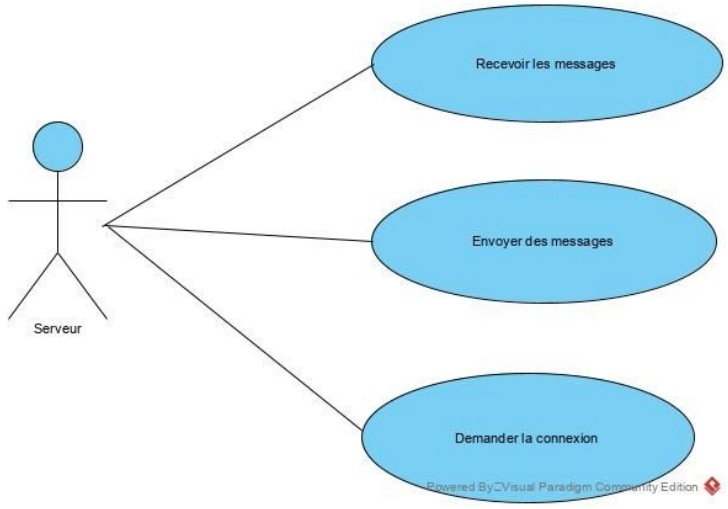
Titre : Effet de cartes

Description :
En tant que joueur
Je souhaite pouvoir activer les effets de mes cartes
Afin de modifier le cours du jeu

Critère d'acceptance :

Toutes les cartes du jeu doivent posséder leurs effets

Use case : Système Client



Pour le Use Case système Client l'acteur doit être le Serveur. Nous avons conclu que le client rend comme service au serveur le fait de demander une connexion sur celui-ci, de lui envoyer des messages ou d'en recevoir, comme par exemple envoyer la ressource qu'il sélectionne quand il tombe sur une face "OU" ou encore quand il souhaite ou non faire une deuxième action etc. Nous pouvons donc simplifier en disant que le client rend comme service au serveur le faire d'Envoyer/Recevoir des messages (pour éviter de dire dans le use case quel message il envoie ou reçoit à chaque fois) et de demander de se connecter.

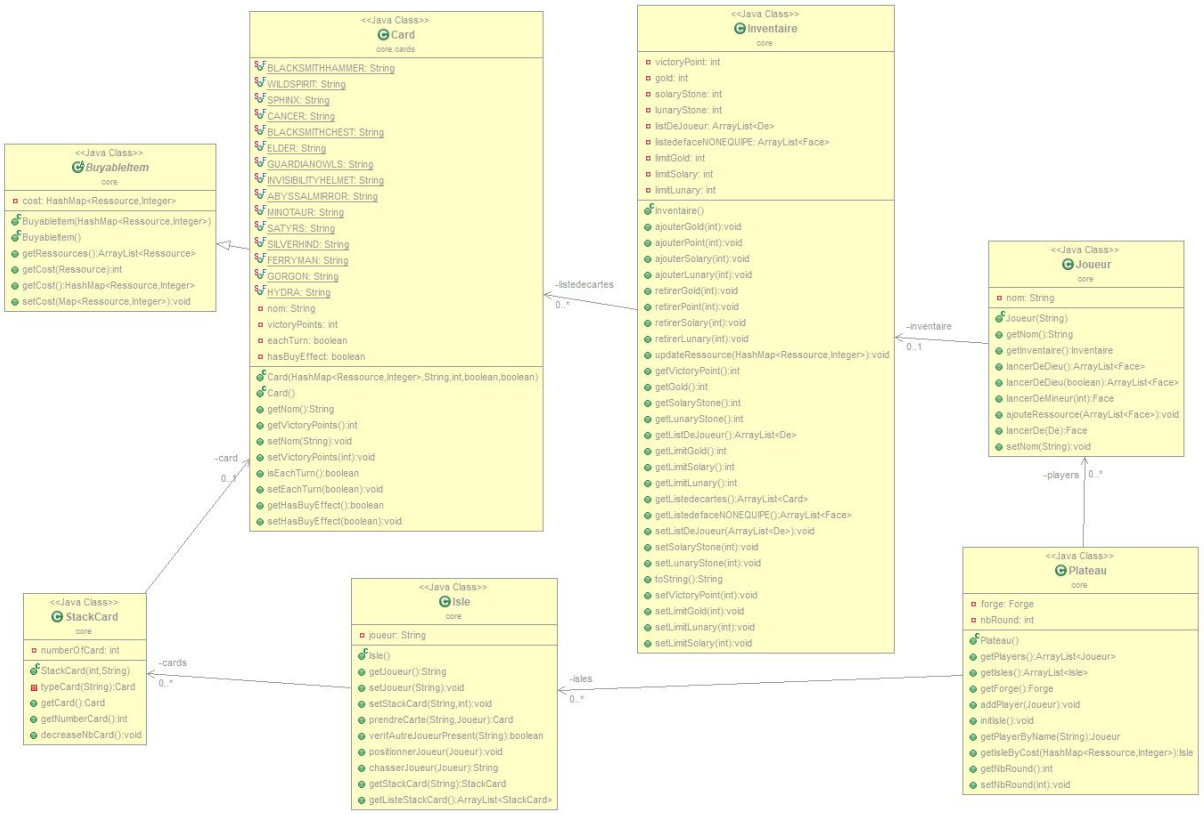
Diagramme de classe

Nos diagrammes de classe sont différents de ceux du premier rendu. En effet, nous avons changé toute l'architecture du projet donc nous avons dû mettre à jour nos différents diagrammes de classe. Nous avons maintenant 4 diagrammes de classe : deux concernant le package "core" que nous avons décidé de séparer pour une question de lisibilité et de compréhension, un pour le package "client" et un pour le package "server".

Diagramme de classe : Package.core “Face/De”

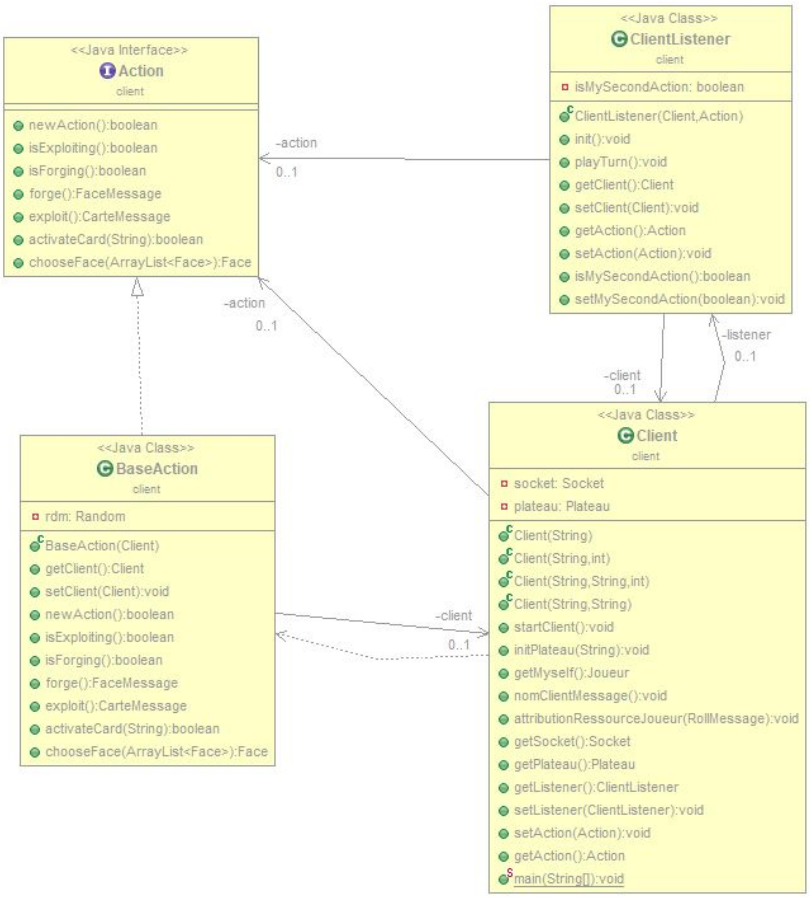
Concernant le premier diagramme de classe du package “core”, nous nous sommes concentré sur les faces et les dés.
Nous avons un plateau contenant les joueurs de la partie.
Chaque joueur possède son inventaire.
Dans l’inventaire du joueur, nous trouvons une liste de dés et une liste de faces non équipées.
Chaque dé contient 6 faces.
Le plateau possède aussi une forge qui permet l’achat des différentes faces.
Une forge comprend plusieurs forgeCategory qui héritent de buyableItem et chage forgeCategory comprend des faces.

Diagramme de classe : Package.core “Carte”



Le deuxième diagramme de classe du package “core” traite des cartes.
Le plateau contient toujours les joueurs et ils ont toujours un inventaire.
Un inventaire comprend une liste de cartes achetées par le joueur.
Le plateau possède des îles.
Chaque île possède une stackCard (pile de cartes) contenant des cartes.
Une carte est un buyableItem.

Diagramme de classe : Package.Client

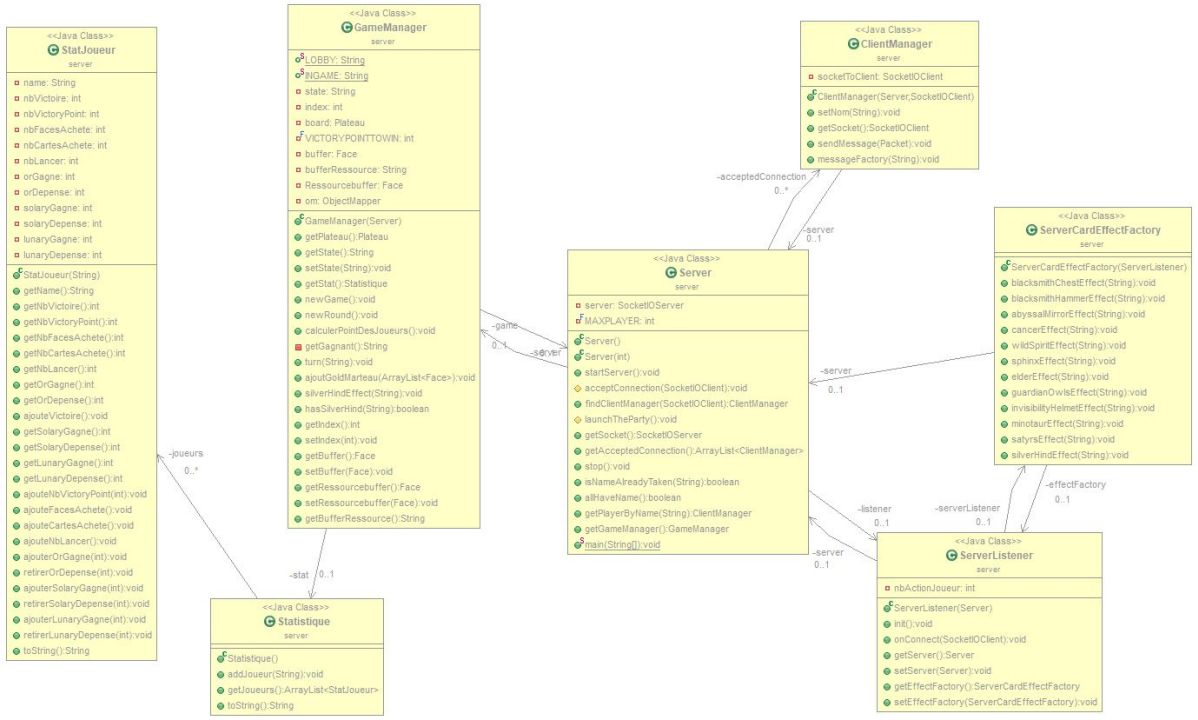


Nous avons décidé de séparer le client et le serveur dans 2 packages différents. Concernant le package “client”, nous avons créé deux classes et une interface :

- une classe ClientListener où sont placé toutes les receptions de messages
- une interface qui fait le lien entre la classe BaseAction et la classe Client

Un client possède un listener et un listener possède un client.

Diagramme de classe : Package.Server



Enfin, pour le package “server” nous avons aussi séparer les réceptions de messages du serveur. Le calcul des statistiques se fait du côté du serveur, avec la classe Statistique qui comprend des StatJoueur.. Nous avons aussi créé une classe ServerCardEffectFactory qui sert à appliquer les effets des différentes cartes et de les communiquer aux clients d’où son lien avec le server et le serverlistener.
Le GameManager qui s’occupe du déroulement de la partie comprend un serveur et les statistiques.
Le serveur à la liste des joueurs connectés, les écouteurs et le GameManager.

Diagrammes de séquence

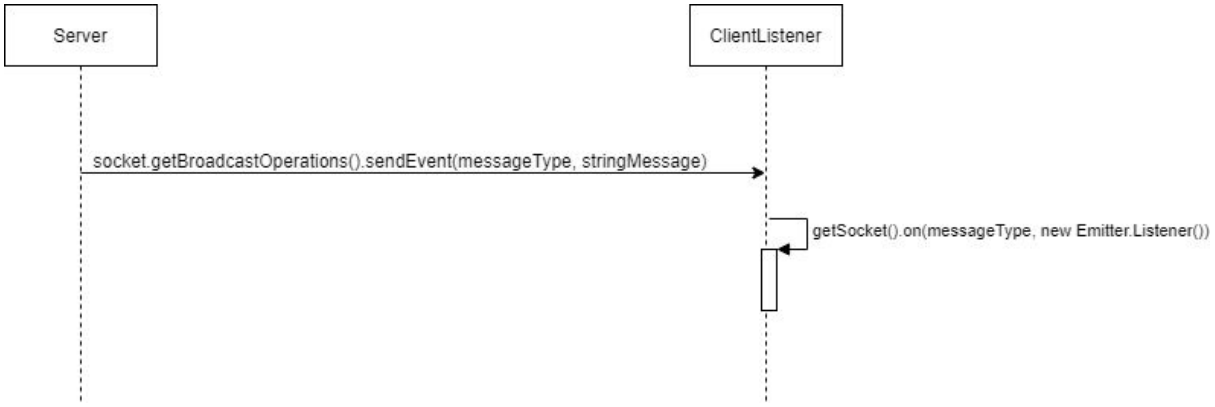


Diagramme de séquence : Recevoir les messages du serveur

Le serveur envoie un message au client avec comme paramètre le type de message et un string Message contenant les données du message. Le client le réceptionne et exécute l'action en fonction.

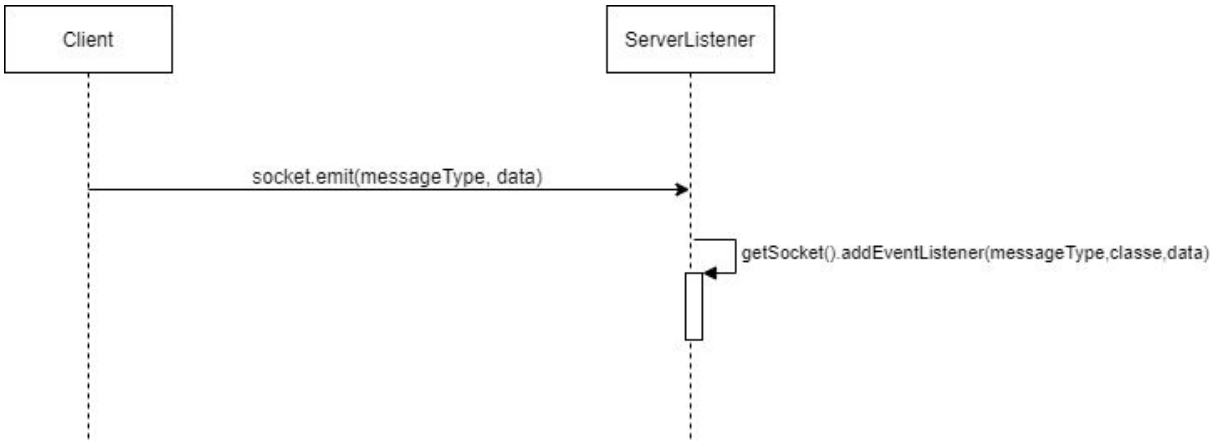


Diagramme de séquence : Envoyer les messages au serveur

Le client crée une socket avec comme paramètre un string correspondant au type du message, et le message envoyer. Le serveur va réceptionner le message et exécuter l'action en fonction.

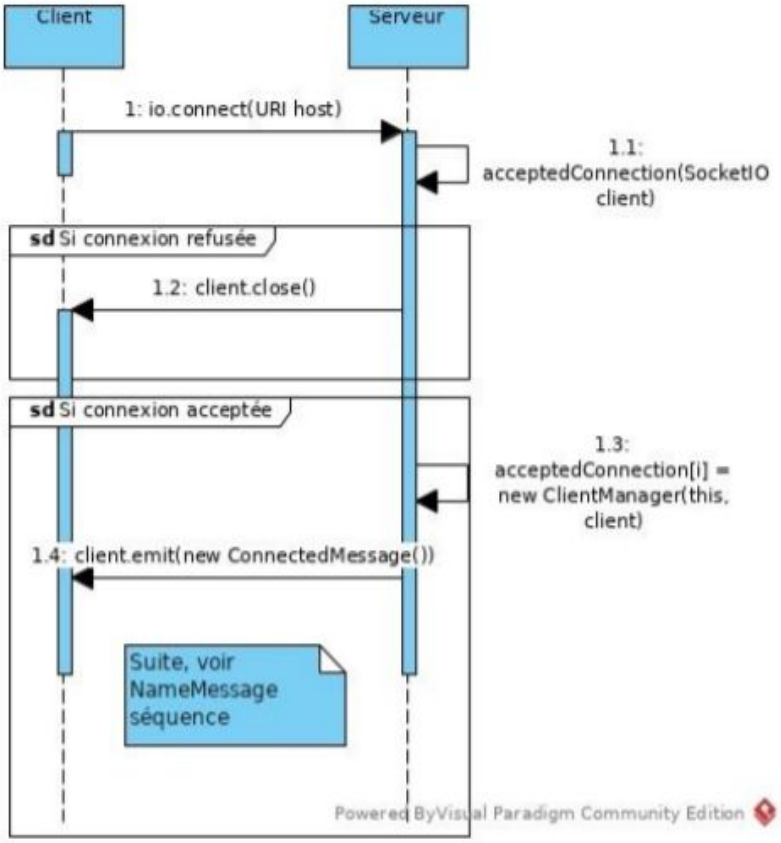


Diagramme de séquence : Se connecter

Message envoyé par le serveur pour notifier à un client que sa connexion a été accepté et qu'il a été ajouté dans la partie
Le message est vide, le serveur envoie juste un événement de type ConnectedMessage

Le client se connecte au serveur, le serveur regarde si les conditions sont remplies pour l'accepter. Sinon il close le socket vers le client. Si oui, il lui alloue un ClientManager et lui envoie un ConnectedMessage pour lui notifier qu'il a été accepté et qu'il attend qu'il lui donne son pseudo.

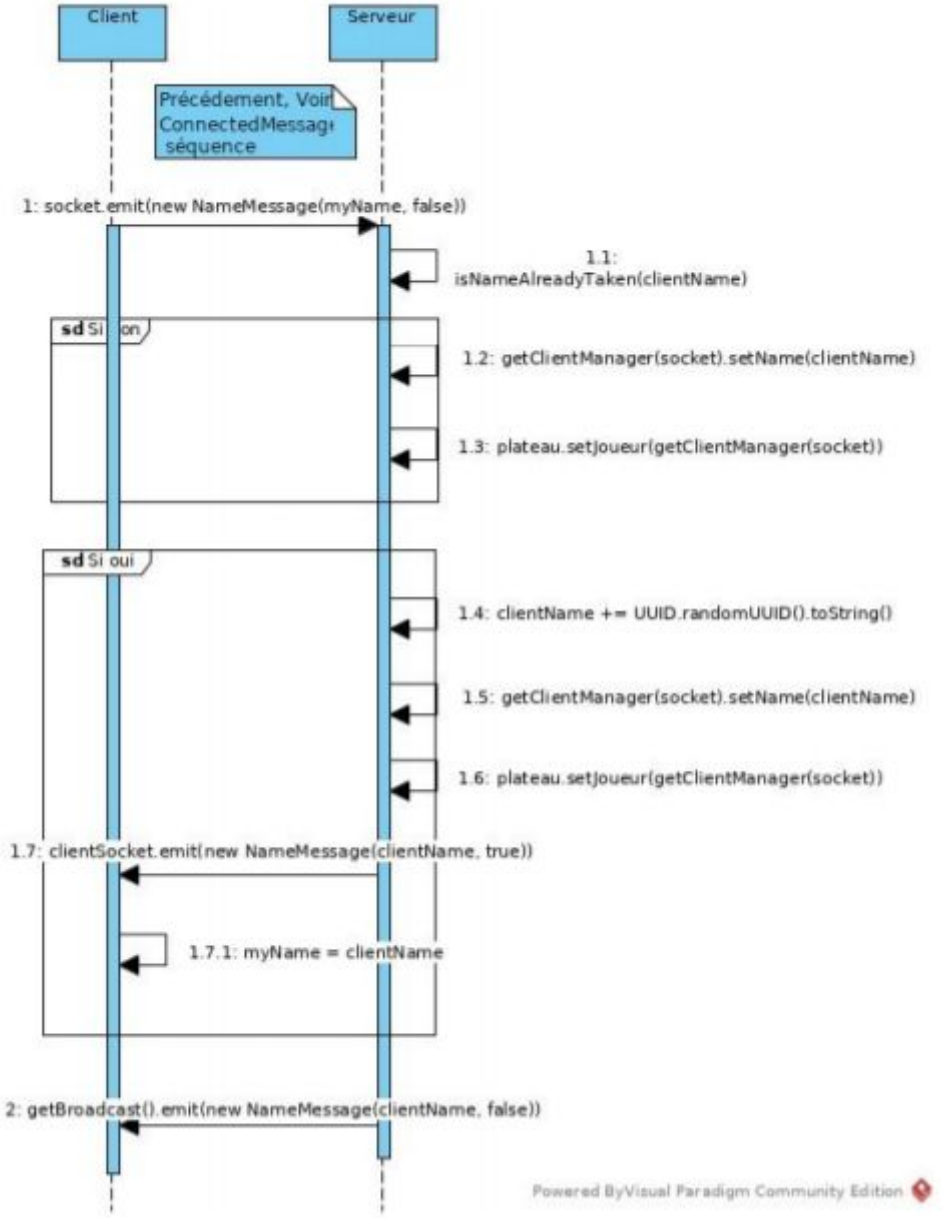


Diagramme de séquence : choisir nom

Message envoyé par le Client au Serveur afin que le Client puisse donner son pseudo au Serveur une fois la connexion acceptée. Si le pseudo donné par le Client est déjà utilisé par un autre Client, le serveur modifie le nom du Client et lui renvoie.
Le message contient les informations suivantes :

- Le nom du joueur
- un boolean qui indique si le serveur a dû modifier le nom du joueur car il était déjà pris.

Le Client envoie son pseudo au serveur, le serveur regarde si le nom n'est pas déjà utilisé par un autre client. Si non il set le nom du joueur dans son clientManager et il le rajoute dans le plateau. Si oui il rajoute un UUID derrière le pseudo du joueur, il set le nom et le rajoute dans le plateau.

Puis envoie un NameMessage au client avec son pseudo modifié et la valeur du boolean en true, pour indiquer que le pseudo a été modifié.
Puis il envoie en broadcast le nom du joueur qui vient de se connecter.

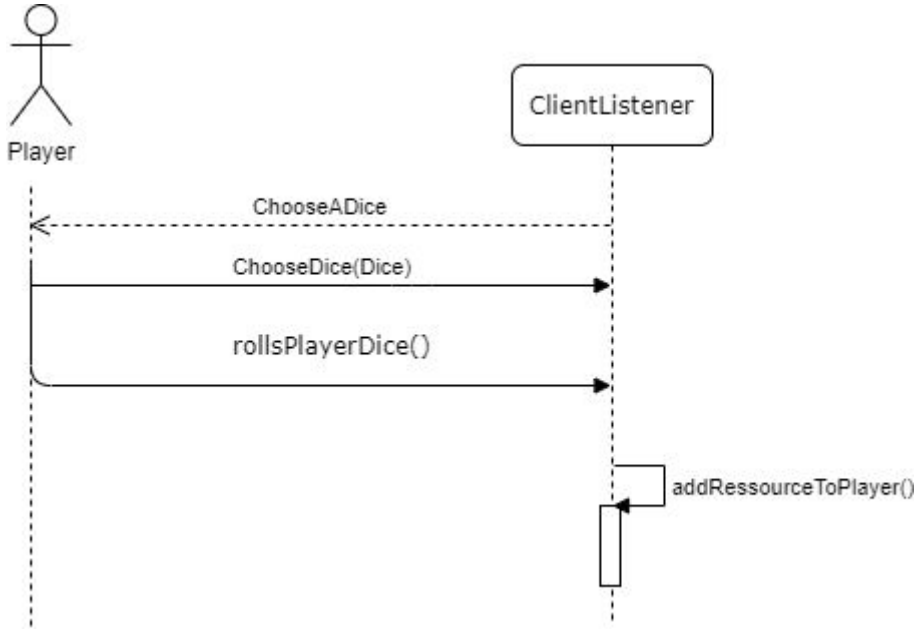


Diagramme de séquence : Lancer un Dé

Lorsque le joueur aura la possibilité de lancer un seul dé (lancer de dé mineur grâce à un effet par exemple), on lui proposera alors de choisir le dé qu'il veut lancer, parmis ceux proposés.
Le joueur choisit alors son dé et le lance, les ressources de la face obtenues lui seront alors ajoutés à son inventaire.

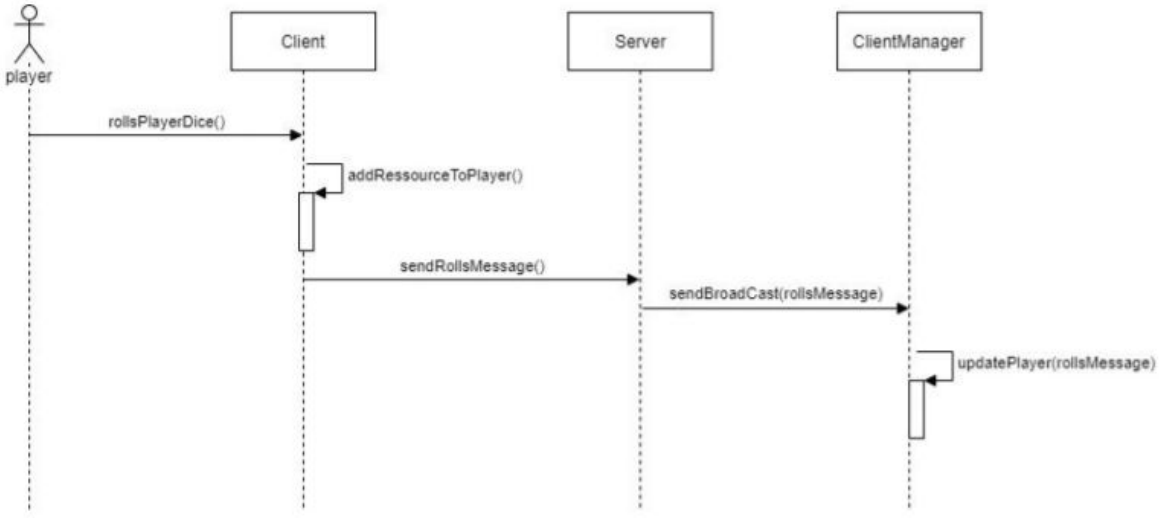


Diagramme de séquence : Lancer les Dés

Lorsque le joueur peut lancer ses 2 dés (début de chaque tour ou grâce à un effet),
celui-ci va les lancer puis récoltera les ressources des faces qu'il aura obtenu.
Un message est alors envoyé au serveur, puis au clientManager pour mettre à jour les données : le joueur a bien lancer ses dés et a obtenu des ressources.

Diagramme de séquence : Faire une nouvelle action et payer 2 solary

Lorsque que le joueur a déjà effectué une action (forge ou exploit), le système va alors vérifier si celui-ci possède au minimum 2 solary.
Si c'est le cas, le jeu proposera au joueur qu'il veut effectuer une nouvelle action en dépensant 2 solary. Si le joueur accepte, ses ressources diminuent de 2 solary et il pourra ensuite effectuer une deuxième action, sinon il aura fini son tour.
De la même façon, si le joueur n'a pas au moins 2 solary en sa possession, son tour se termine alors automatiquement pour laisser le joueur suivant jouer.

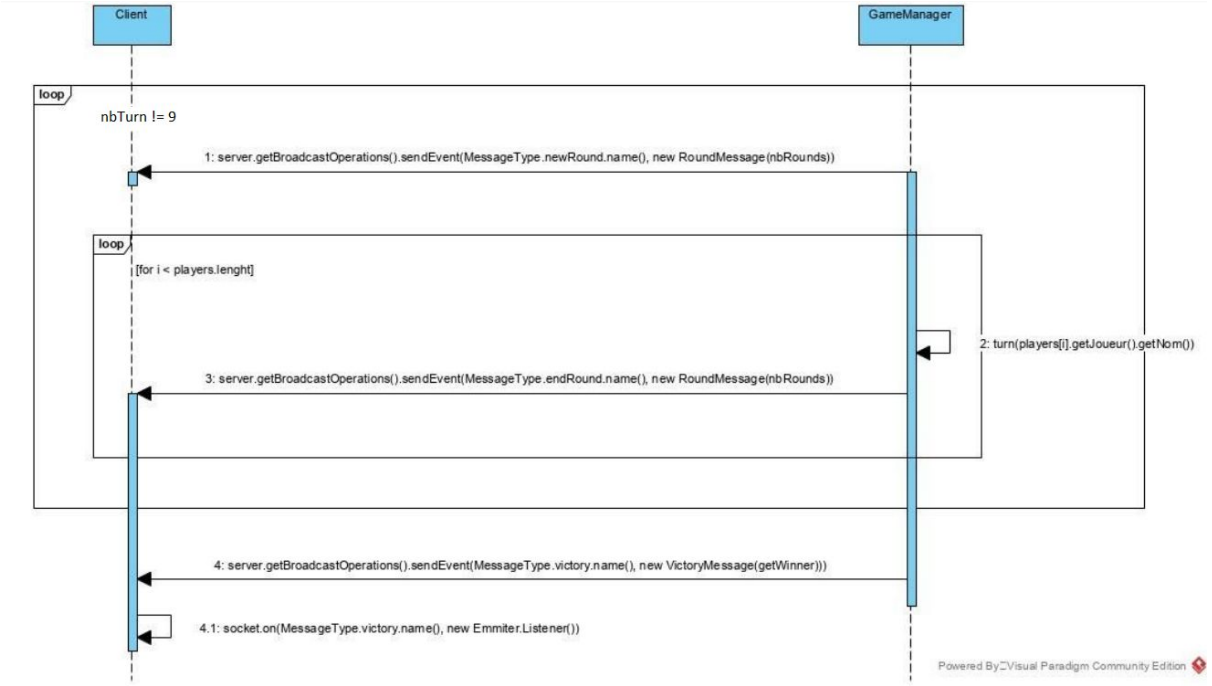


Diagramme de séquence : Déclarer le gagnant

Dans la première version du code, où le joueur continuait de jouer tant qu'il n'avait pas gagner avec un certains nombre de points défini, le joueur continuera de jouer pendant 9 tours.
Le jeu prendra fin lorsque le dernier joueur à jouer jouera son 9e tour.

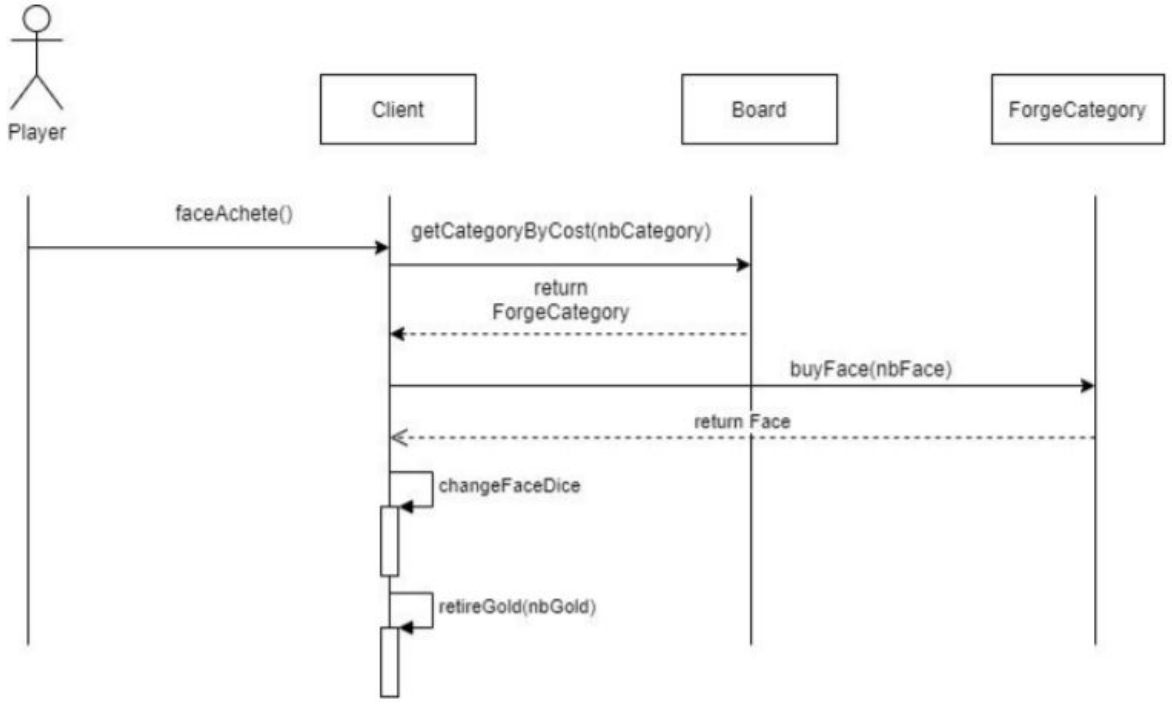


Diagramme de séquence : Forge

Lorsque le joueur choisi de forge l'un de ses dés, il lui sera alors proposé les faces disponibles dans le forge. Puis, le joueur choisira la face qu'il souhaite récupérer pour la mettre sur l'un de ses dés.
La face sera alors changée et retirée de la forge, puis le joueur se verra retirer des golds selon le coup de la face qu'il a choisie.

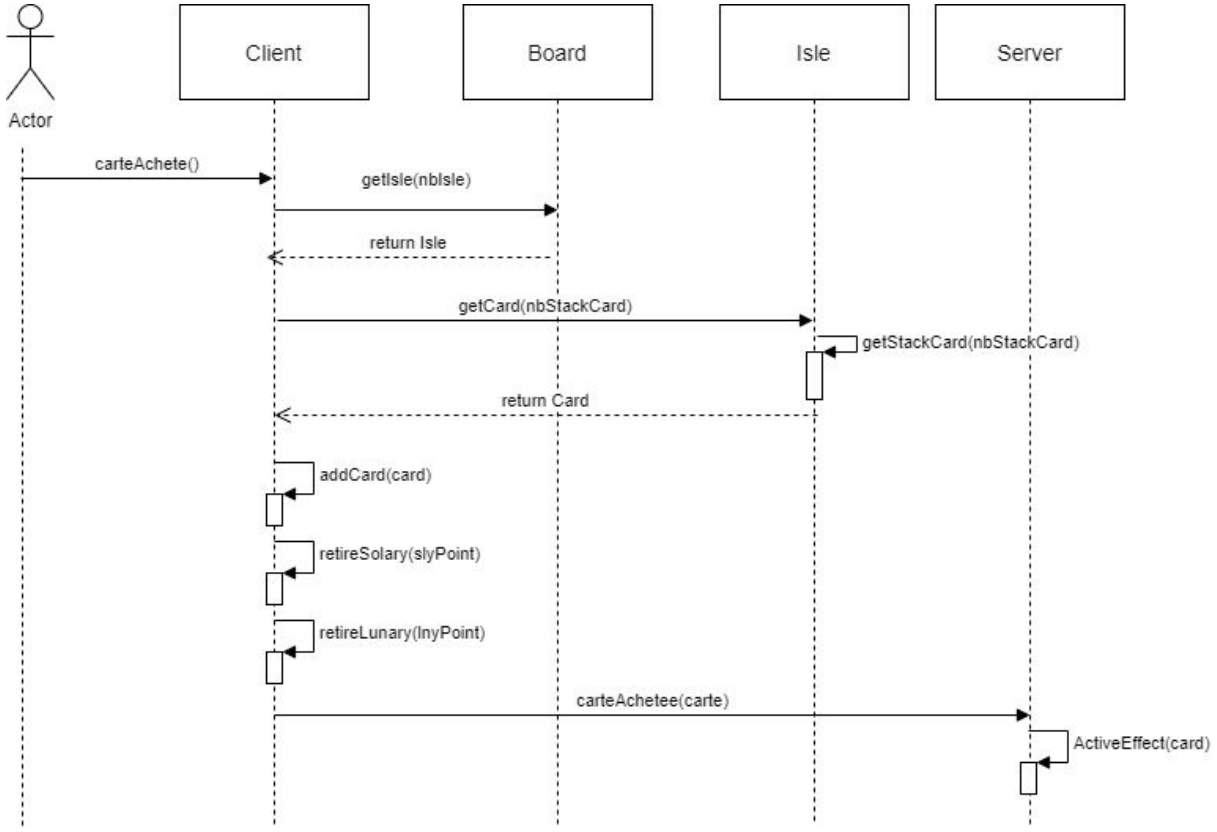


Diagramme de séquence : Choisir l'exploit et acheter une carte

Pour l'exploit, rien n'a changé. lorsque le joueur décide de faire un exploit plutôt que de forger l'un de ses dés, nous passons à ce moment là, à l'achat d'une carte.
Lors de l'achat d'une carte, le début reste le même que lors de la dernière itération. Lorsque le joueur décide d'acheter une carte et choisi sa carte (si elle est disponible), la carte lui sera alors attribuée par le jeu (addCard) et des ressources retirés en fonction du prix de la carte.
Ce qui change pour cette partie, c'est l'ajout des effets de la carte. Depuis l'itération 3, le effets des cartes ont été implémenter et donc après l'achat de la carte en dépensant les ressources, des effets vont alors s'activer. Les effets sont soit immédiat, soit des effets permanents qui s'activeront au prochain tour et pour tous les tours suivant.

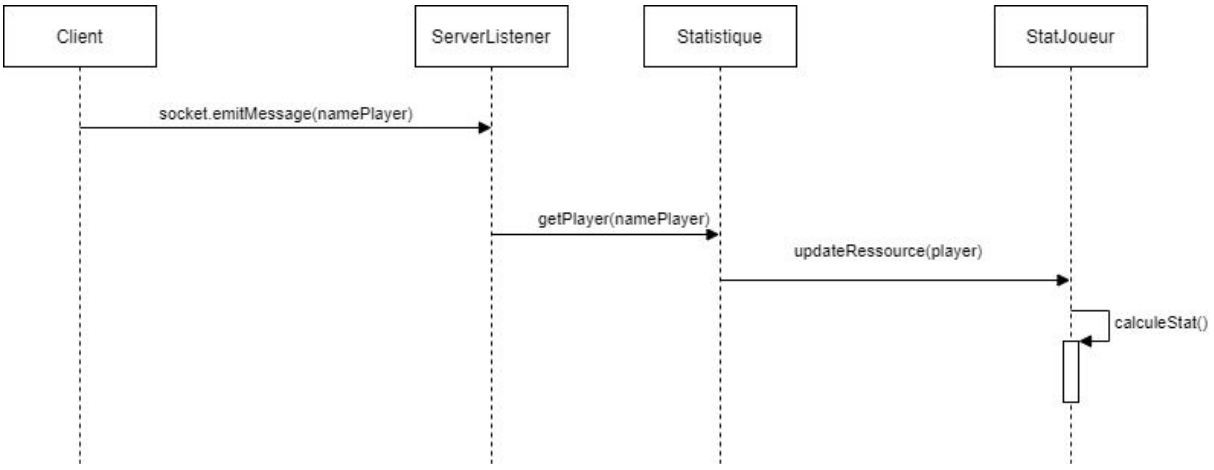


Diagramme de séquence : Obtenir les statistiques

Le client envoi un message de type marteau, lorsque le ServerListener le reçoit il renvoi le joueur vers la classe Statistiques. Celle ci va appeler la classe StatJoueur pour lui calculer ses statistiques.

Interaction entre le/les joueurs et le moteur

Protocole d'échange

CarteMessage / HuntMessage

CarteMessage qui notifie le serveur puis les autres joueurs que le joueur qui joue à acheter une carte

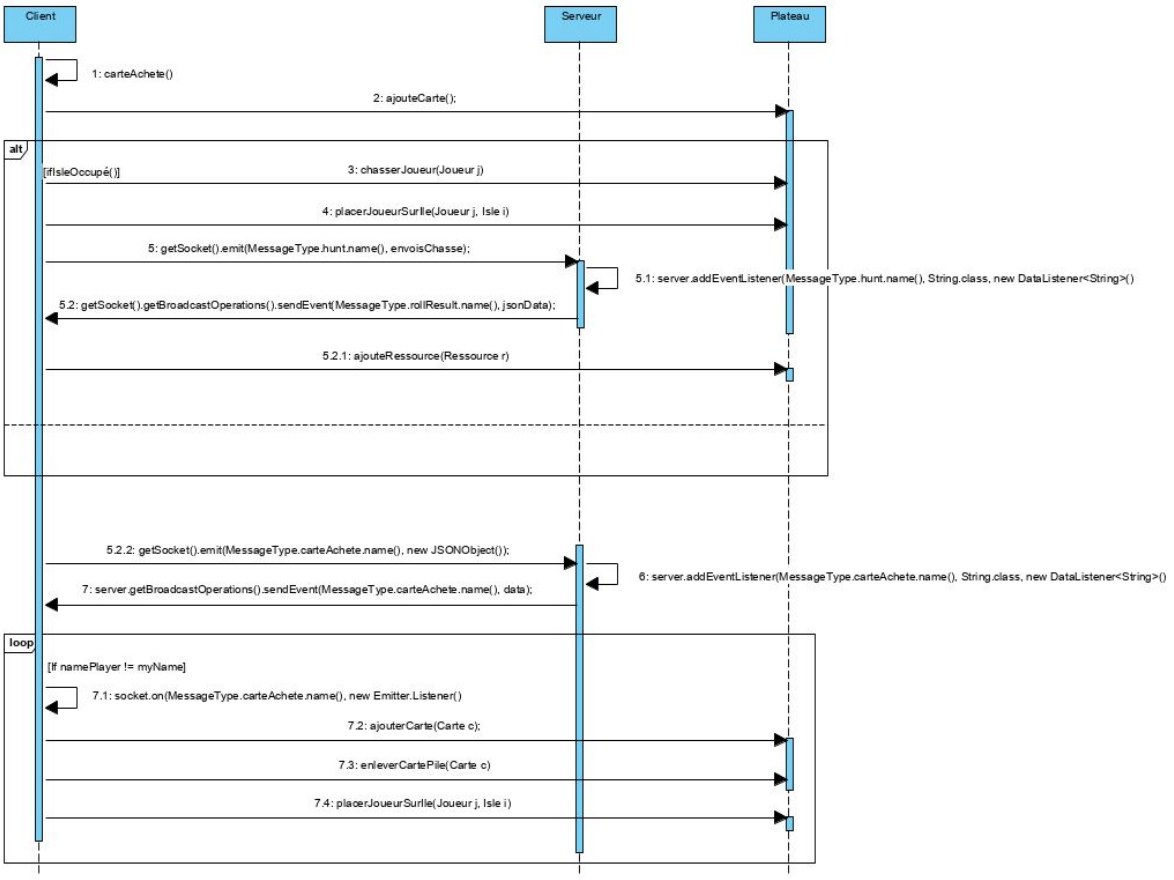
Le CarteMessage contient les informations suivantes

- Le nom du joueur
- La carte achetée
- Le nombre de lunary stone nécessaire à son achat
- Le nombre de solary stone nécessaire à son achat
- Le numéro de l'île sur laquelle elle a été acheté

HuntMessage est envoyé par le client au serveur puis transmis aux autres clients. Le client envoie ce message quand il a du chassé un autre client pour acheter une carte. À la réception de ce message le serveur lance les dés du joueur chassé et transmet le résultat du lancer à tous les joueurs

Le HuntMessage contient les informations suivantes :

- Le nom du joueur chassé



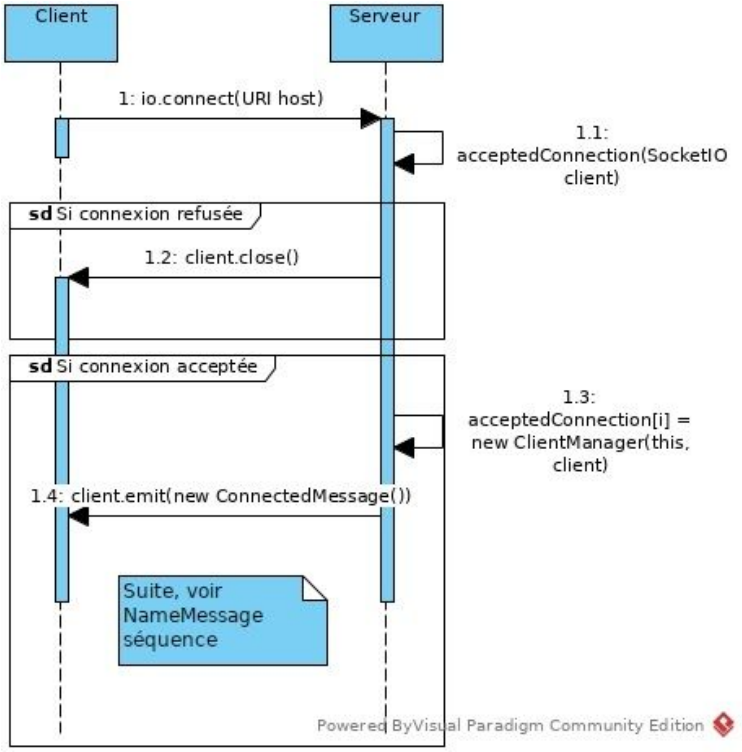
Le client choisit une carte à acheter puis il va la communiquer au plateau pour l'acheter. Si l'île où la carte est acheté est occupé par un joueur, alors le joueur ayant acheté la carte va chasser le joueur positionner sur l'île. L'acheteur est ensuite positionné sur l'île. un message HuntMessage est envoyé vers le serveur qui va roll les dés du joueur chassé. Le résultat du roll est ensuite retourner au joueur chassé qui va ajouter les ressources à son inventaire.

Après l'achat de la carte, le client envoie un CarteMessage vers le serveur qui va ensuite l'envoyer en broadcast aux joueurs. Après la réception, les joueurs qui ne sont pas le joueur qui a acheté la carte, vont ajouter la carte dans l'inventaire du joueur l'ayant acheter, enlever la carte acheté de la pile du plateau et placer le joueur sur l'île.

ConnectedMessage

Message envoyé par le serveur pour notifier à un client que sa connexion a été accepté et qu'il a été ajouté dans la partie

Le message est vide, le serveur envoie juste un événement de type ConnectedMessage



Le client se connecte au serveur, le serveur regarde si les conditions sont remplies pour l'accepter.

Si non il close le socket vers le client.

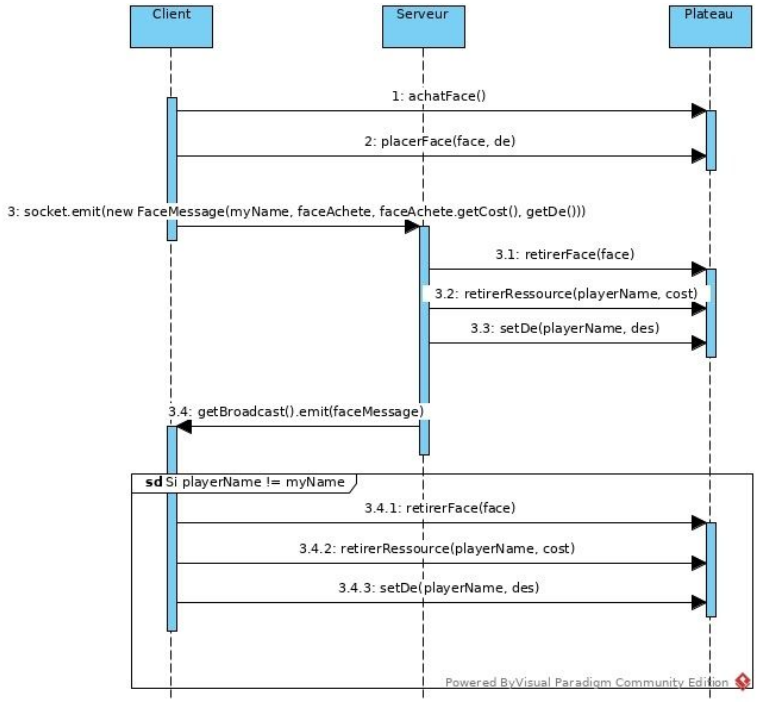
Si oui, il lui alloue un ClientManager et lui envoie un ConnectedMessage pour lui notifier qu'il a été accepté et qu'il attend qu'il lui donne son pseudo.

FaceMessage

Message envoyé par le client pour notifier le serveur puis les autres joueurs qu'il a acheté une Face et qu'il l'a placé sur un de ses dés.

Le message contient les informations suivantes :

- Le nom du joueur
- La face achetée
- Le coût en ressource de la face
- Les dés modifiés du joueur



Le Client achète une face, il place la face sur un de ses dés. Il envoie au serveur un FaceMessage avec son nom, la face, le coût de la face, et ses dés modifiés.

Le serveur retire la face de la forge, il retire les ressources du coût au joueur, puis il actualise les dés avec les dés dans le FaceMessage.

Puis il broadcast le FaceMessage aux clients. Si le client qui reçoit le message a un nom différent que dans le FaceMessage alors il retire la face du message dans la forge, il enlève les ressources du coût au joueur du message et il actualise les dés du joueur.

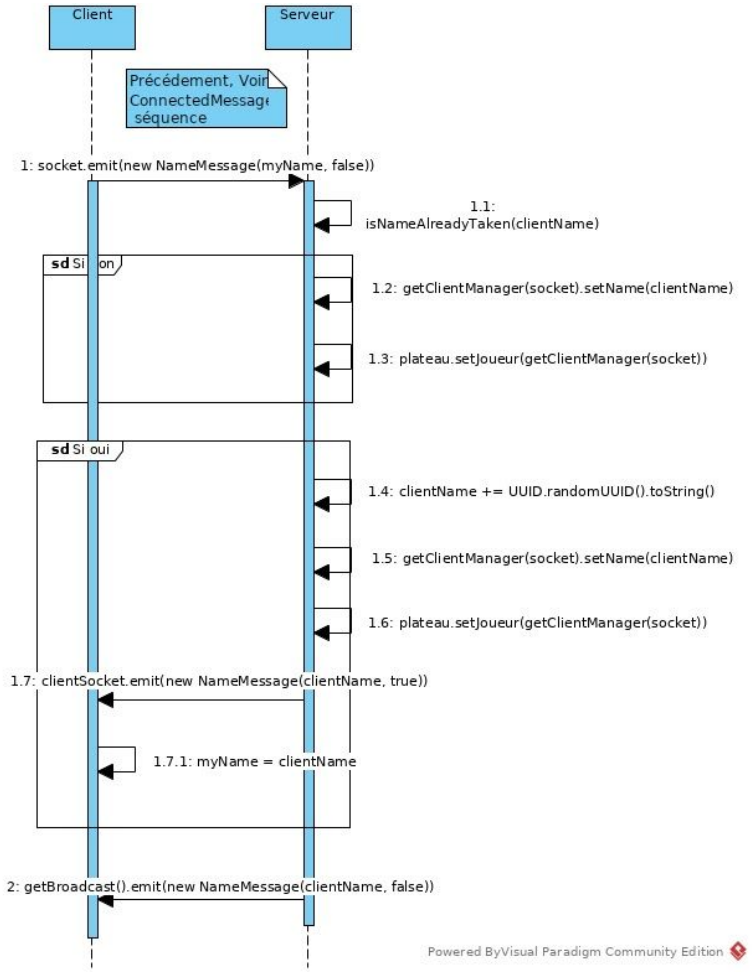
NameMessage

Message envoyé par le Client au Serveur afin que le Client puisse donner son pseudo au Serveur une fois la connexion acceptée.

Si le pseudo donné par le Client est déjà utilisé par un autre Client, le serveur modifie le nom du Client et lui renvoie.

Le message contient les informations suivantes :

- Le nom du joueur
- un boolean qui indique si le serveur a dû modifier le nom du joueur car il était déjà pris.



Le Client envoie son pseudo au serveur, le serveur regarde si le nom n'est pas déjà utilisé par un autre client.

Si non il set le nom du joueur dans son clientManager et il le rajoute dans le plateau.

Si oui il rajoute un UUID derrière le pseudo du joueur, il set le nom et le rajoute dans le plateau. Puis envoie un NameMessage au client avec son pseudo modifié et la valeur du boolean en true, pour indiquer que le pseudo a été modifié.

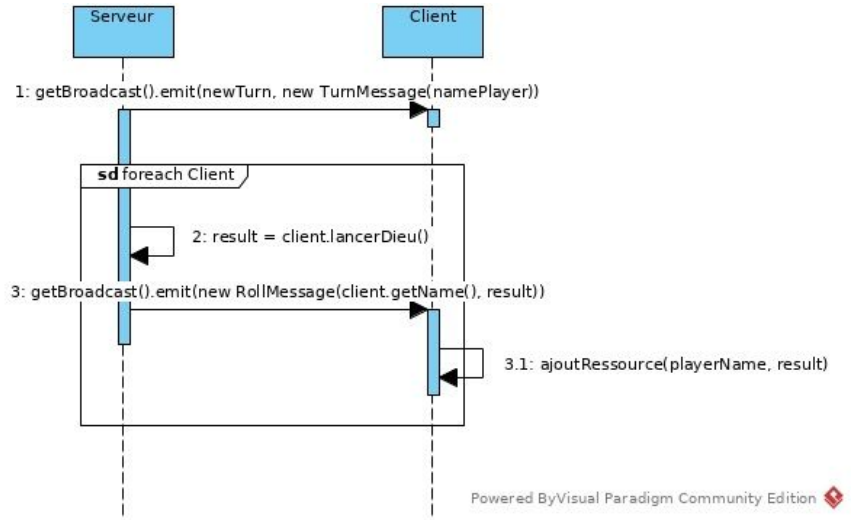
Puis il envoie en broadcast le nom du joueur qui vient de se connecter.

RollMessage

Message envoyé par le Serveur aux Clients afin de leur donner le résultat des lancers de dés.

Le message contient les informations suivantes

- Le nom du joueur à qui est attribué le lancer
- Les faces sur lesquelles il est tombé



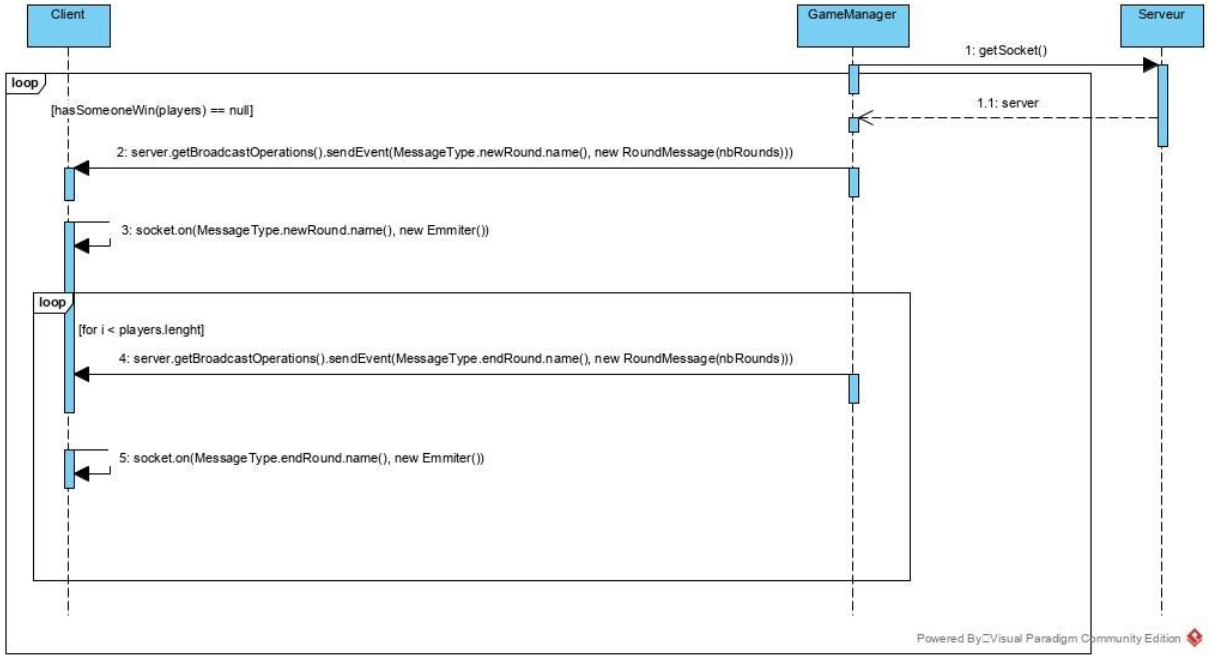
Le serveur broadcast le début d'un nouveau tour en désignant le joueur qui doit jouer. Ensuite pour chaque Client il lance les dés et envoie les résultats en Broadcast avec le nom du joueur et le résultat du lancer. Quand le client reçoit ce message il ajoute les ressources des faces au joueur associé.

RoundMessage

Message envoyé par le serveur aux Clients pour leur indiquer une nouvelle manche

Le message contient les informations suivantes

- Le numéro de manche



Le GameManager broadcast à chaque nouveau tour le numéro de manche tant qu'il n'y a pas de gagnant. Une fois que les joueurs ont joué, le GameManager envoie à chaque joueur un message pour dire que la manche est terminée.

TurnMessage

Message envoyé par le serveur aux Clients pour leur indiquer un nouveau et qui est le joueur qui doit jouer.

Le message contient les informations suivantes

- Le nom du joueur qui doit jouer

Durant la manche, les joueurs reçoivent un message de la part du GameManager qui leurs précise quel joueur est en train de jouer.

VictoryMessage

Message envoyé par le serveur aux Clients pour leur indiquer le gagnant de la partie.

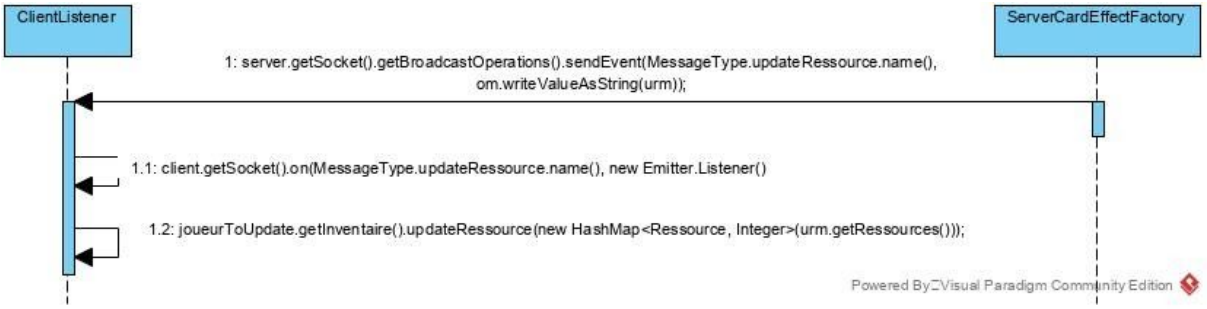
Le message contient les informations suivantes

- Le nom du joueur qui a gagné

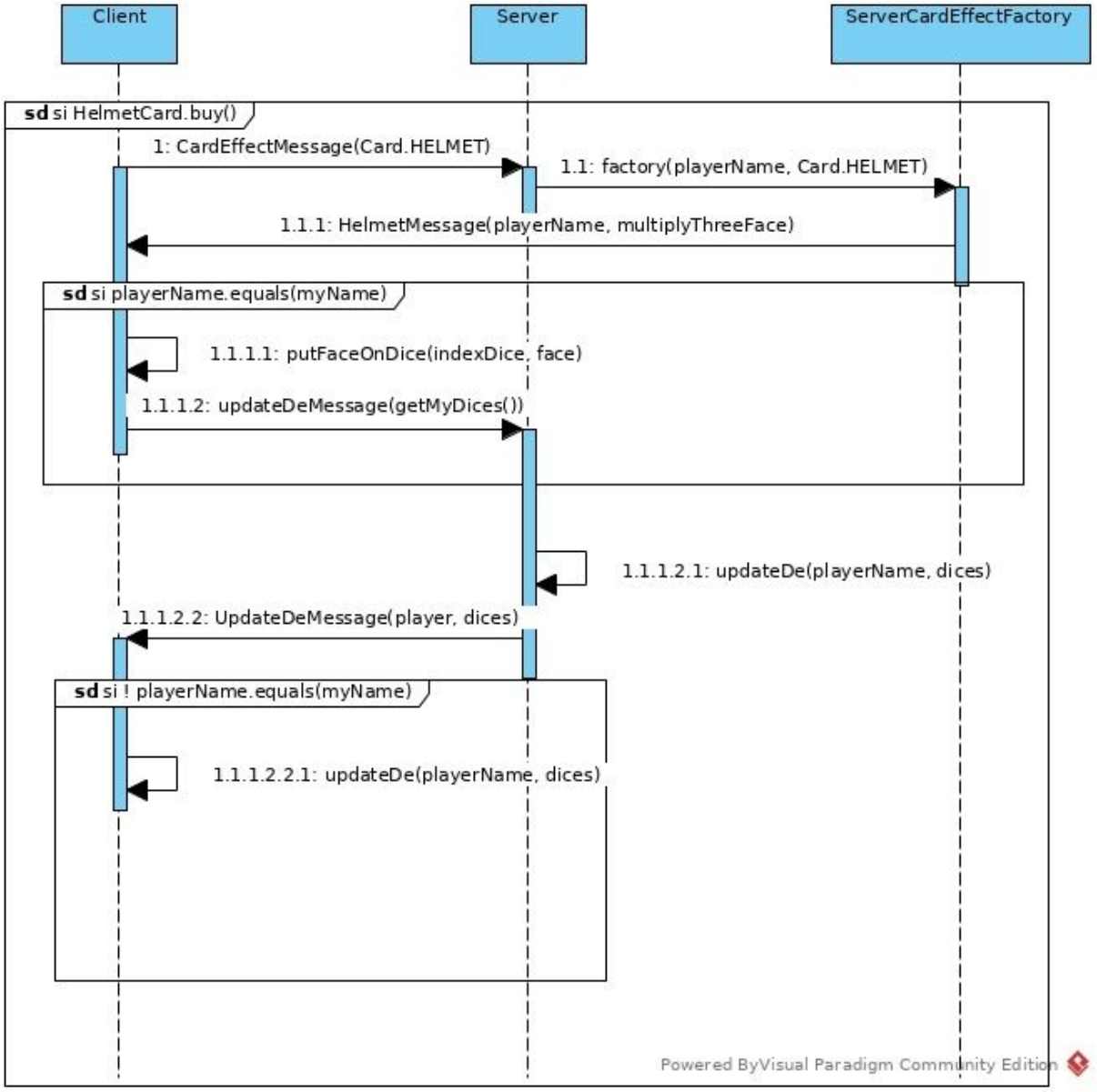
Une fois qu'un joueur a atteint le nombre de Victory point nécessaire à la victoire, le GameManager envoie à chaque client quel joueur à gagner et ils l'affichent.

UpdateRessourceMessage

Ce message est utilisé dans le serverCardEffectFactory lors de plusieurs activation d'effet de carte.
Ce message permet de mettre à jour les ressources d'un joueur dans le plateau des autres en précisant le nom du joueur qui doit avoir ses ressources à jour et les ressources qu'on va modifier.



UpdateDeMessage/HelmetMessage

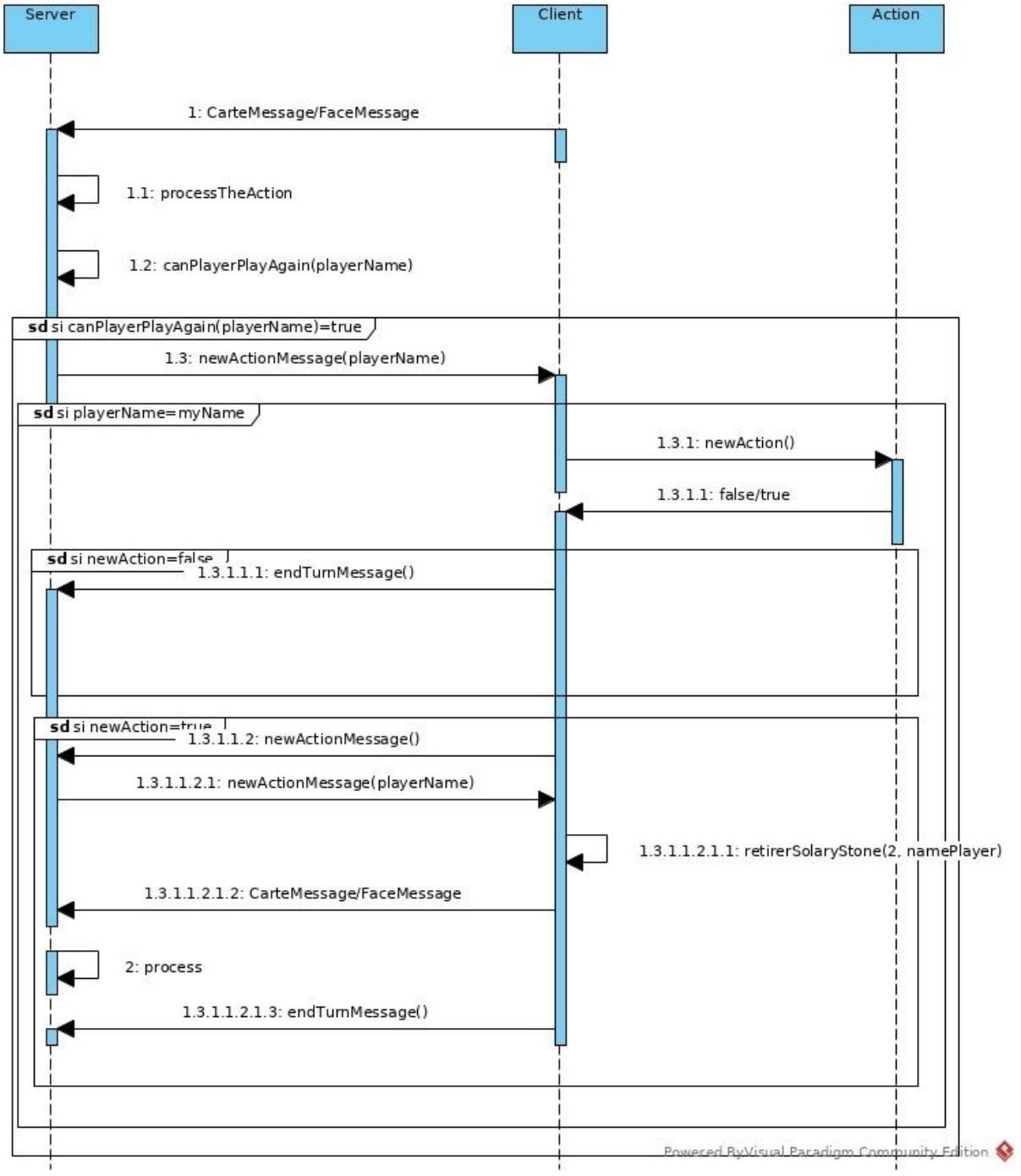


Quand un client achète la carte HelmetCard, il envoie un CardEffectMessage au serveur, qui le met dans sa factory.
La factory envoie au client la face multiplyThree, le client choisit sur quelle dé et quelle face mette la face multiplyThree.
Une fois placée le joueur renvoie un message au serveur avec sa liste de dé mis à jour. Le serveur met à jour les dés des joueurs et renvoie le message aux autres joueurs afin qu'ils puissent eux aussi mettre la liste des dés du joueur à jour.

SatyrsMessage

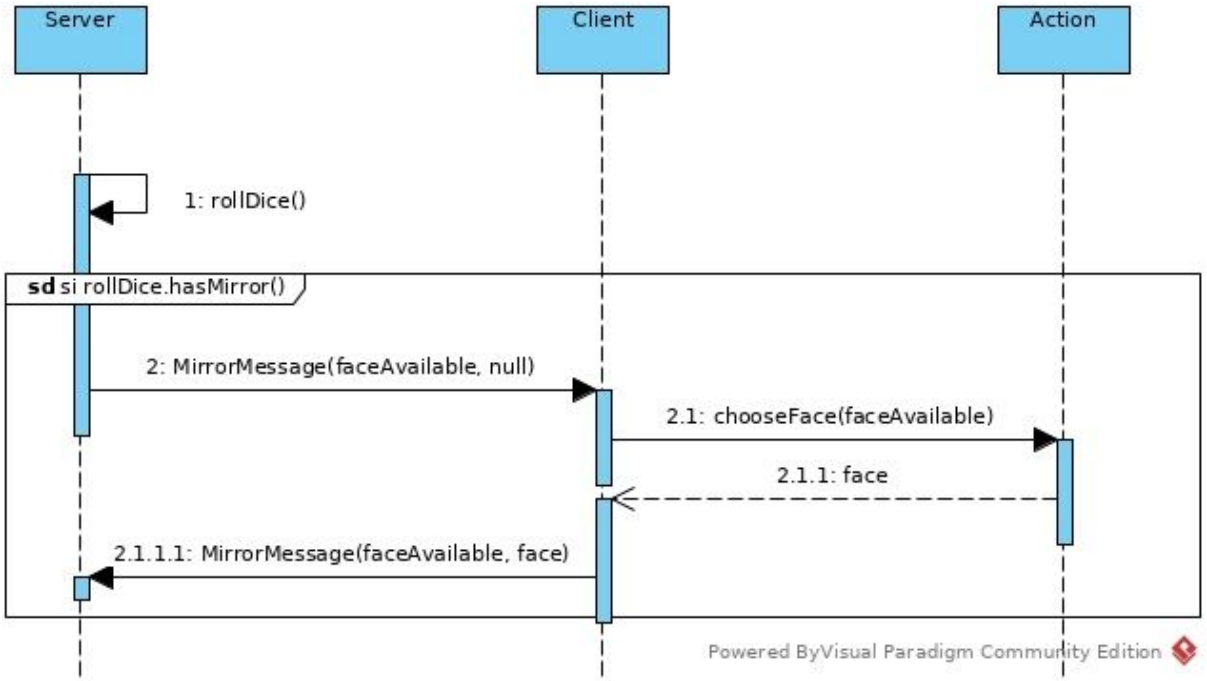
Ce message est utilisé pour l'effet Satyrs dans le ServerCardEffectFactory. Il va servir à récupérer toutes les faces obtenues quand tous les autres joueurs vont rolls leurs dés et permettre ensuite au joueur dont c'est le tour, de choisir 2 des faces parmis toutes celles obtenues. Un message est donc envoyé au ClientListener avec le nom du joueur qui a acheté la carte et les rolls des joueurs. Dans le ClientListener, nous aurons alors le choix des faces par le joueur, puis nous enverrons un message permettant de mettre à jour les ressources.

NewActionMessage



Quand le joueur effectue sa premiere action (achat de carte ou de face), le serveur regarde si le joueur peut rejouer. S'il peut rejouer il envoie au client un NewActionMessage.
Le client décide s'il souhaite rejouer ou non.
S'il ne souhaite pas rejouer il envoie un EndTurnMessage qui met fin à son tour et fait débuter le prochain.
S'il souhaite rejouer il envoie un NewActionMessage au serveur, le serveur va retransmettre ce message à tout le monde afin que tout le monde puisse retirer 2 pierres solaire à l'inventaire du joueur.
Enfin le client envoie sa deuxième action puis un EndTurnMessage.

MirroirMessage



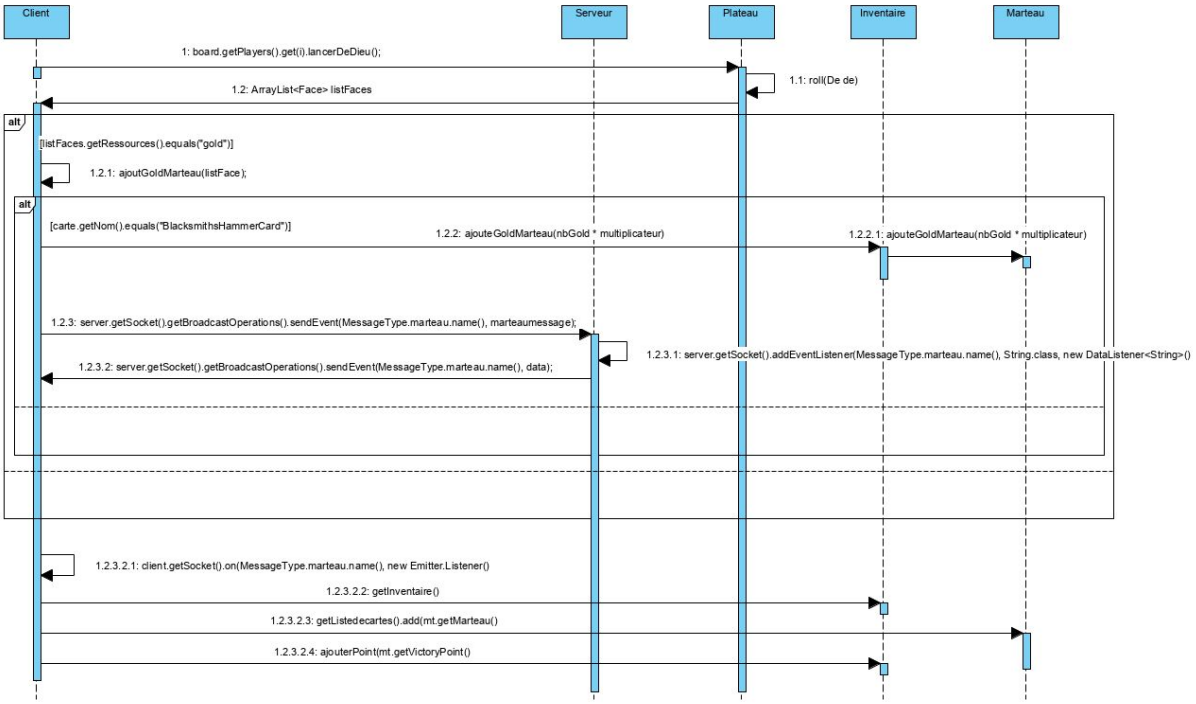
Quand le serveur roll les dés, si un joueur tombe sur un mirroir, le serveur lui envoie un MirroirMessage avec les faces qu'il peut choisir (face qui ont été pris dans le roll des autres joueurs).
Le client choisit la face qu'il souhaite possèder et renvoie un MirroirMessage avec les faces qui lui étaient disponibles et la face selectionnée.

MarteauMessage

Le MarteauMessage contient les informations suivantes :

- une carte marteau
- le nom du joueur possédant la carte
- un nombre de victory point gagné grâce au marteau

Le client, après avoir lancé les dés, si il possède une carte marteau et qu'il a obtenu des ressources de type "or", il peut choisir d'en mettre sur son marteau.
Après avoir ajouté de l'or, il envoit la modification de son marteau au serveur qui va ensuite le transmettre aux autres joueurs.
A la réception, les joueurs vont mettre à jour la carte du joueur possédant le marteau et vont lui ajouter des victory points si le marteau en a fait gagné.



MarteauAcheteMessage

Le MarteauAcheteMessage contient les informations suivantes :

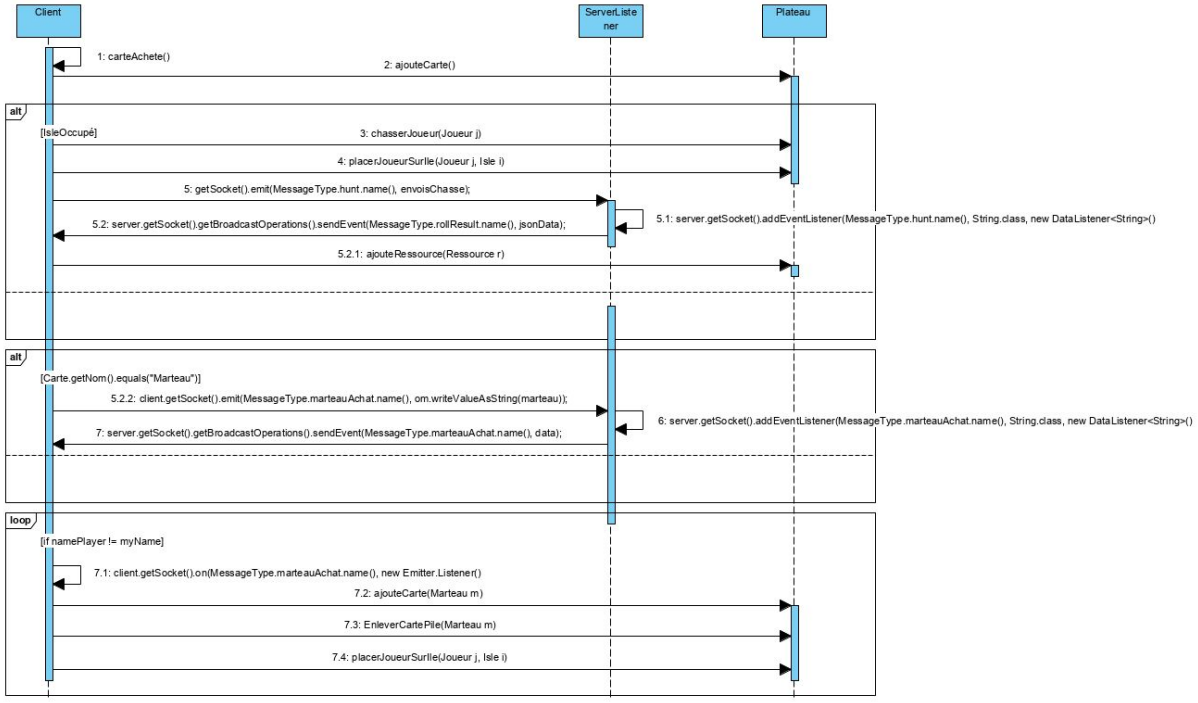
- Le nom du joueur
- La carte marteau achetée
- Le nombre de lunary stone nécessaire à son achat
- Le nombre de solary stone nécessaire à son achat
- Le numéro de l'île sur laquelle elle a été acheté

Le HuntMessage est utilisé de la même façon que pour le diagramme de séquence CarteAchete.

Le client choisit une carte à acheter puis il va la communiquer au plateau pour l'acheter. Si l'île où la carte est acheté est occupé par un joueur, alors le joueur ayant acheté la carte va chasser le joueur positionner sur l'île. L'acheteur est ensuite positionné sur l'île. un message HuntMessage est envoyé vers le serveur qui va roll les dés du joueur chassé. Le résultat du roll est ensuite retourner au joueur chassé qui va ajouter les ressources à son inventaire.

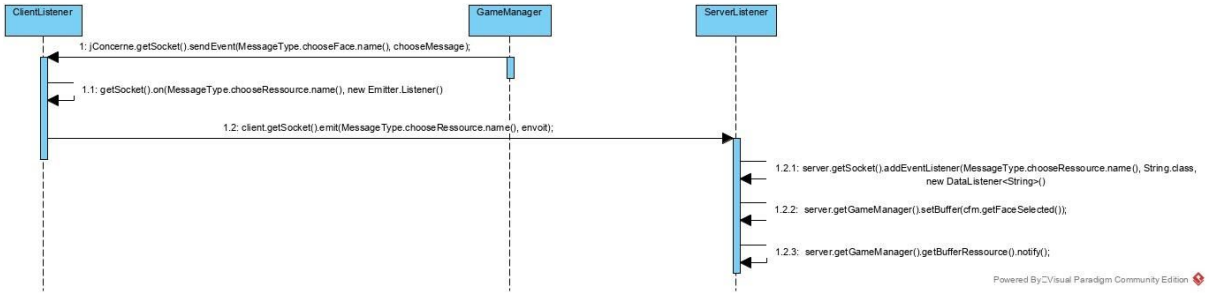
Après l'achat de la carte, le client va vérifier si la carte acheté est une carte marteau ou non. Si c'est le cas, alors il envoie un MarteauAcheteMessage vers le serveur qui va ensuite l'envoyer en broadcast aux joueurs. Après la réception, les joueurs qui ne sont pas le joueur qui a acheté la carte, vont ajouter la carte dans l'inventaire du joueur l'ayant acheter, enlever la carte acheté de la pile du

plateau et placer le joueur sur l'île.



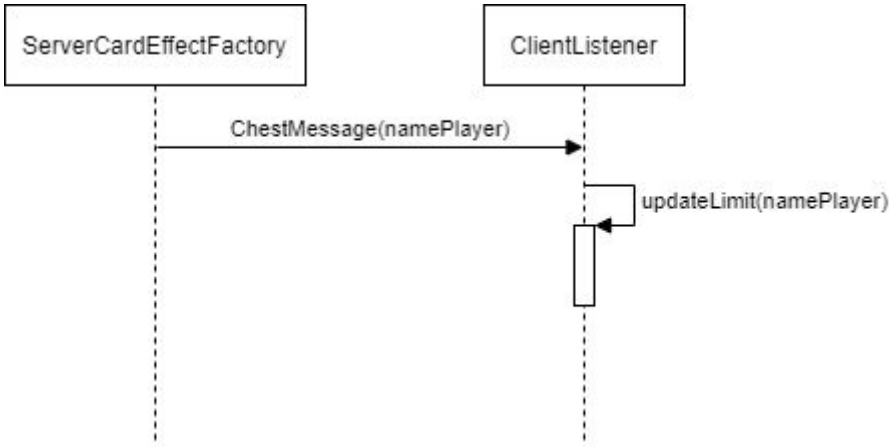
ChooseFaceMessage

Le message ChooseFaceMessage est utilisé quand le joueur obtient une face de type "OU". On lui demande demande alors de choisir parmis les 2 ressources qui sont sur la face et celui-ci nous renvoie la ressource qu'il a choisis et nous mettons à jour côté serveur son choix dans son roll



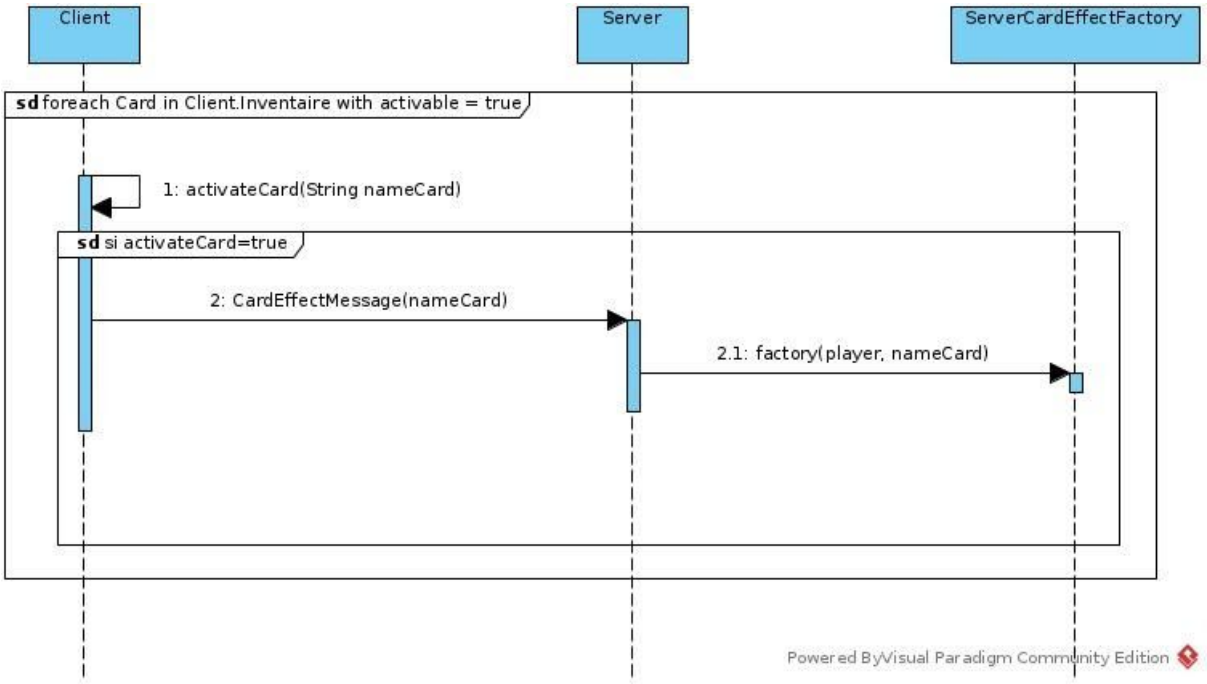
ChestMessage

Le ChestMessage est utilisé lorque le joueur achète la carte Chest, le ServerCardEffectFactory envoi un ChestMessage avec le nom du joueur qui à acheter la carte. Lorsqu'il le reçoit, le client met à jour les limit des golds, des pierres solaires et des pierres lunaires.



Lorsque le joueur achète la carte Chest, le ServerCardEffectFactory envoi un ChestMessage avec le nom du joueur qui à acheter la carte. Lorsqu'il le reçoit, le client met à jour les limit des golds, des pierres solaires et des pierres lunaires.

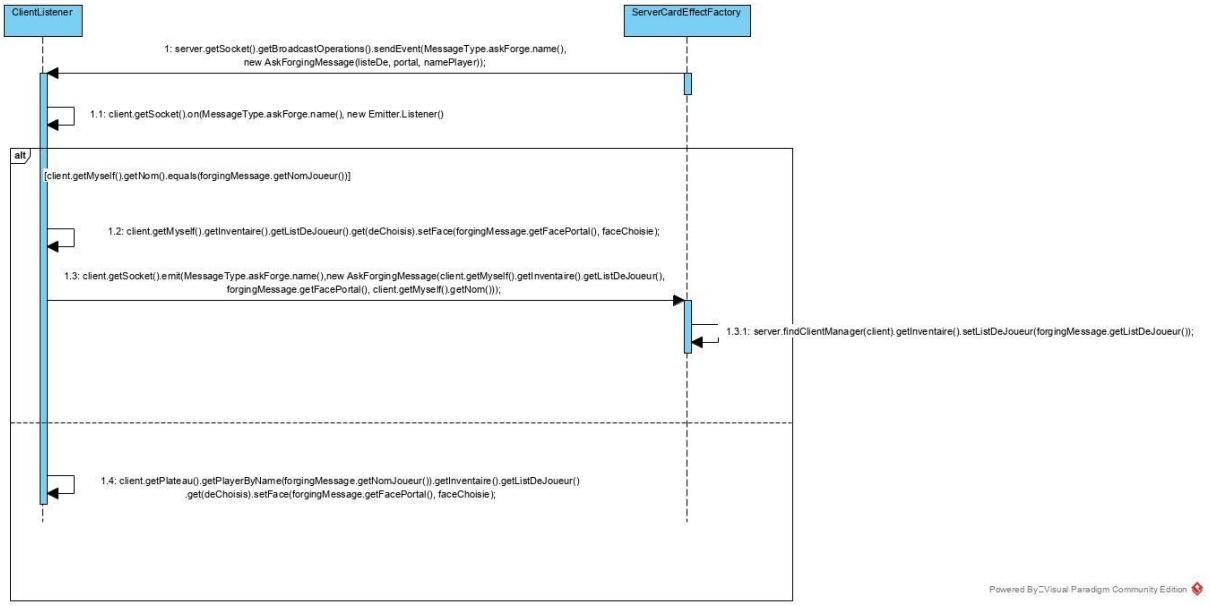
CardEffectMessage



Pour chaque Carte dans son inventaire qui sont activables à chaque tour, le joueur réfléchit à s'il souhaite la jouer ou pas, s'il souhaite la jouer un envoie un CardEffectMessage avec le nom de la carte dans le message. Une fois reçu le client envoie à sa factory d'effet le nom de la carte et l'effet.

AskForgingMessage

Ce message est utilisé lorque qu'un joueur achète et utilises la carte Mirroir Abyssal qui lui permet de forger sur un de ses dés la face portal. On envoie donc en broadcast le message comme quoi tel joueur a joué la carte Mirroir abyssal puis du côté du CLient Listener si le joueur qui reçoit le message est le joueur concerné alors il fait parvenir au serveur sur quel face de quel dé il veut forgé le portail puis le serveur mettra à jour. Les autres joueurs quant à eux , mettront simplement à jour leur plateau avec la modification.



Conclusion

Analyse de votre solution

Point fort

- Synchronisation des tours et des actions des joueurs.
- Synchronisation des plateaux des joueurs et du serveur.
- Interface Action permettant de modifier “facilement” le comportement du joueur.
- ServerCardEffectFactory qui permet de facilement modifier les effets des cartes et d’en rajouter.
- Plateau dans le serveur qui permet d’ajouter la possibilité de voir si un joueur triche ou pas sur ces actions en vérifiant ses ressources, les faces disponibles, les cartes disponibles, les cartes qu’il possède et les faces qu’il possède.
- Plateau dans les clients qui leurs permettent de savoir en local l’état du plateau et de ne pas demander à chaque tour l’état du plateau au serveur.

Point faible

- Quand le serveur a besoin d’une action de la part du client (choisir une des ressources sur une face, choisir une face parmis une liste grâce au mirroir), le serveur a des problèmes pour envoyer la demande et l’intégrer dans la suite de son code.

Évolution Effectuée

En terme d’évolution nous avons rajouté l’interface Action et la classe BaseAction. Qui permet de séparer le comportement du bot de la classe ClientListener.
Nous avons créer la CardEffectFactory et la ServerCardEffectFactory, qui permet de modifier d’ajouter des effets des cartes “facilement”.

Évolution prévue

Dans les futurs itérations nous avons prévu d’ajouter plus de bots, pour cela nous allons créer de nouvelles classes qui extends de Action et créer des Clients qui utilisent ces nouvelles classes pour décider de leurs actions. La classe BaseAction va quant à elle, être mis au placard, car c’était notre bot “random” qui effectuait toutes ses actions en random pour que nous puissions tester la réponse du client et du serveur en attendant de créer des comportements plus stratégiques.