EPSI

C# FINAL PROJECT : DIAMOND

Thomas Delecroix

Développement d'une solution applicative en langage C#

Version: 1.0

Sommaire

1 Règles

1.1 Triche

Est considéré comme triche tout partage de code, fichier, test, outil de test ou script. La triche est strictement interdite et entraînerait une note nulle sur le rendu en cours et potentiellement des pénalités sur les autres rendus sans négociation possible.

1.2 Deadline

La deadline (date et heure de rendu) est stricte, aucun rendu ne sera accepté après la deadline spécifiée en amont du sujet ce qui entraînera une note nulle.

1.3 Soumission

Il est demandé de ne soumettre que les fichiers demandés dans l'énoncer. Les fichiers binaires et fichier de configurations sont à supprimer avant envoie sous peine de réduction de note sauf mention contraire du sujet.

1.4 Encodage

Tous les fichiers envoyés doivent être encodés en ASCII, UTF-8 ou UTF-16. Tout autre type d'encodage est interdit et sera considéré comme illisible.

1.5 Bibliothèque

Seules les bibliothèques spécifiées explicitement dans le sujet sont autorisées, toutes les autres sont interdites et leur utilisation implique une note nulle.

1.6 Output

Quand des exemples visuels sous un certain format sont donnés, ils sont à suivre scrupuleusement au caractère près.

2 Rendu

2.1 Date

La date de rendu est définie à 23h42 le 13/03/2022.

2.2 Git

Le rendu se fera via Git. Il est recommandé d'utiliser Github pour héberger ce dernier.

Pour la correction il est nécessaire de donner les accès de lecture au profil suivant : Tetragg.

2.3 Build

Le projet étant un projet Client / Serveur, il est demandé qu'il soit fourni avec deux sous-projets et qu'il puisse produire deux binaires :

- diamant_client : pour tout le côté client ;
- diamant_server : pour tout le côté serveur.

2.4 Bibliothèque

Les bibliothèques autorisées sont :

L'ajout de bibliothèques supplémentaires autorisées peut être demandé au professeur.

Toutes les autres sont interdites d'utilisation.

Note

Il est possible de demander l'autorisation d'utilisation d'autres bibliothèque par mail au professeur.

3 Diamond

Diamant est un jeu de plateau pour 3 à 8 joueurs. Le but est simple, réunir le plus de diamants possible. Pour cela, il va falloir explorer des grottes à la recherche de cette précieuse pierre.

3.1 Préambule

3.1.1 YAML

YAML est un format de fichier qui est utilisé dans l'informatique pour décrire des entités. Ce format a pour but d'être simple (moins verbeux) en lecture et manipulation humaine, plus que ses concurrents XML ou JSON.

Voici un exemple :

```
- martin:
    name: Martin
    job: Developer
    age: 10
    skills:
      - python
      - perl
      - pascal
- tabitha:
    name: Tabitha
    job: Designer
    age: 50
    skills: ~
```

On définit ici deux personnes, Martin et Tabitha, ainsi que des propriétés qui leur sont rattachées. L'utilisation de listes, nuls, strings, entiers ou autres est possible.

Néanmoins on ne fait que définir des objets, en aucun cas on ne peut faire du scripting avec du YAML.

3.2 Règles

3.2.1 Matériel

Dans le jeu, il existe différents types de cartes :

- Trésor : contient un certain nombre de diamants, les joueurs se partagent le butin ;
- Trophée : contient un certain nombre de diamants, les joueurs ne peuvent se le partager ;
- Danger : un piège déclenché pour tous les joueurs.

De plus, chaque joueur possède deux cartes : une carte continuer et une carte sortir. Ces deux cartes sont des décisions que le joueur doit prendre à chaque fin de tour : continuer l'exploration ou sortir de la grotte.

Les joueurs possèdent aussi un coffre-fort dans leur camp pour mettre à l'abri leurs découvertes.

3.2.2 Déroulement de la partie

Dans une partie standard, il y a cinq grottes à explorer (manche de jeu).
Au début de chaque manche, les joueurs pénètrent à l'intérieur de la grotte ensemble.

Ensuite débute les tours. Pour chaque tour, une carte est tirée :

- Trésor : les joueurs présents se partagent le montant équitablement et laisse le surplus qui n'est pas divisible ;
- Trophée : les joueurs laissent le trophée sur le chemin car il n'est pas divisible ;
- Danger : les joueurs déclenchent un piège.

Les diamants récupérés par les joueurs sont stockés dans leurs mains tant que l'exploration de la grotte n'est pas terminée.

Chaque joueur doit maintenant choisir entre deux situations : sortir de la grotte et mettre à l'abri ses diamants ou continuer pour obtenir plus de diamants.

Lorsqu'un joueur choisit de rentrer au camp, il récupère les diamants laissés sur le chemin (des tours précédents). Il se les partage avec les éventuels autres joueurs rentrants pendant le même tour. Si un trophée est sur le chemin du retour, le joueur doit obligatoirement rentrer seul pour le récupérer. S'il est accompagné, pour ne pas faire de

jaloux, tous les joueurs doivent le laisser. Une fois rentrés au camp, les joueurs mettent leurs diamants dans le coffre-fort et attendent la fin de la manche. Les trésors récupérés sont retirés des cartes pour la partie.

Lorsque deux pièges du même type sont déclenchés, les joueurs doivent courir et perdent tous les diamants obtenus lors de l'exploration de cette grotte. Seuls les joueurs rentrés au camp sont indemnes. Une carte piège sur les deux est consommée et retirée du jeu pour la partie.

L'aventure se termine lorsque toutes les grottes ont étés explorés.

Le manuel officiel

3.2.3 Implémentation

Dans notre cas, nous allons utiliser une implémentation serveur / client pour simuler le plateau de jeu et les joueurs. Le serveur sera l'entité qui gère tous les joueurs tandis que le client un simple joueur souhaitant jouer.

Le serveur et clients possèdent des fichiers de configuration pour définir les règles de jeu, adresses, port de connexion et détails techniques.

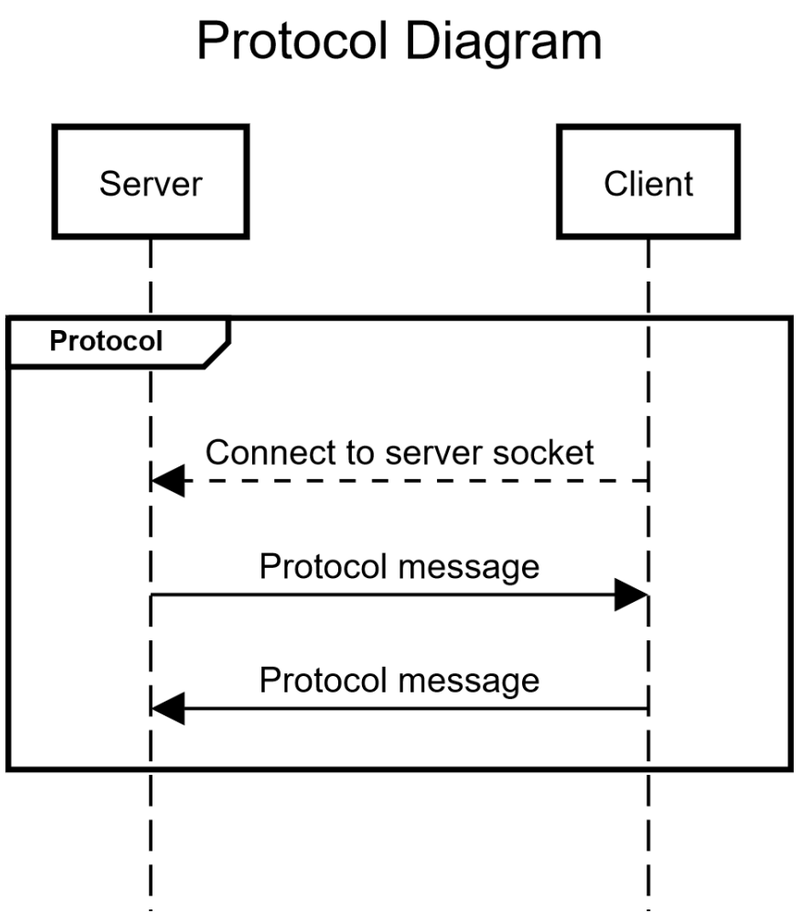
Le plus important point du projet est : quelle que soit l'avancée du projet, il doit rester compatible avec des versions moins avancées. On appelle cela la rétro-compatibilité.

3.3 Diamond Protocol Communication

On définit comme DPC, le protocole de communication pour ce projet. Ce protocole se décline en trois versions. Chacune des versions est une amélioration de la précédente. Quand un protocole est implémenté, il doit rester rétro-compatible avec les précédentes.

Quelle que soit la version du protocole, lors de la connexion du client au serveur, ce dernier donne les protocoles qu'il est capable de gérer. C'est ensuite au client de décider (en fonction des versions qu'il implémente), de laquelle il désire utiliser. Le client va en général choisir l'implémentation la plus évoluée qu'il est capable de supporter.

Dans un premier temps, le client discute avec l'entité principale du serveur pour se mettre d'accord sur le protocole à utiliser.



Côté serveur, la sérialisation du message est toujours une sérialisation objet faite via Jackson. Le message protocole suit toujours le schéma YAML suivant :

```
version: int
encryption: list
compression: list
```

Le chiffrage et la compression sont des listes contenant les différents protocoles supportés. Voici un exemple :

```
version: 3
encryption:
  - aes-128
  - aes-256
  - rsa-1024
  - rsa-2048
compression:
  - gzip
  - deflate
  - br
```

Le client n'a ensuite qu'à répondre simplement comment il souhaite définir la communication :

```
version: int
encryption: string
compression: string
```

```
version: 3
encryption: "aes-256"
compression: "gzip"
```

```
version: 1
encryption: []
compression: []
```

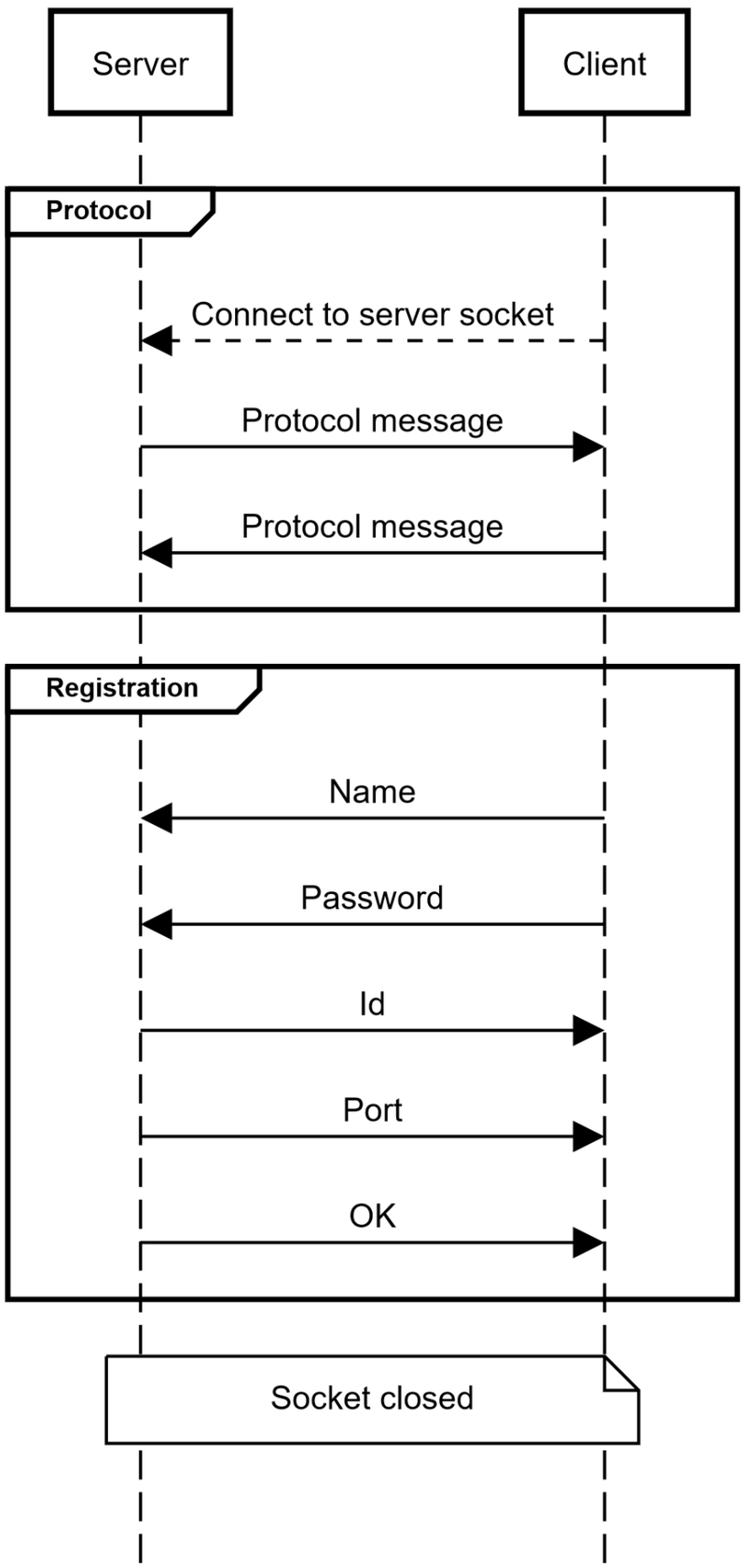
Une fois cela effectué, il va pouvoir s'enregistrer dans une salle de jeu. Le socket de communication pour l'enregistrement reste le même que pour la définition du protocole. Quand l'enregistrement est terminé, les deux parties peuvent fermer le socket.

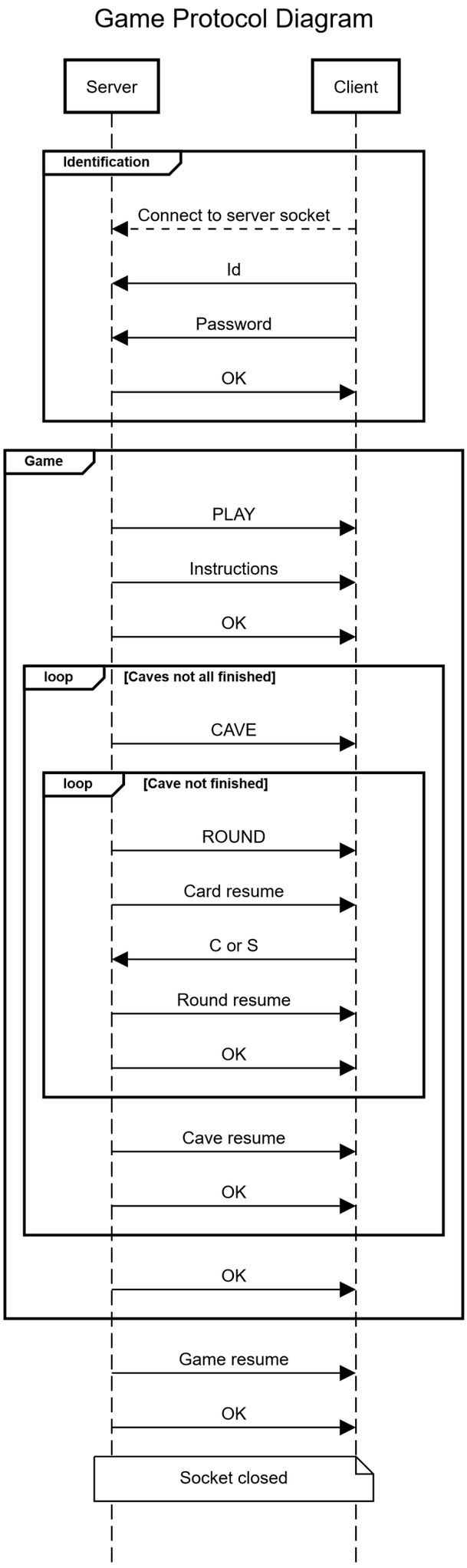
La partie s'effectue sur une autre entité du serveur. À partir des informations données par ce dernier lors de l'enregistrement, on peut accéder directement au serveur de jeu. On s'y connecte en utilisant les informations d'identification et ensuite nous pouvons attendre que tous les joueurs soient prêts pour commencer la partie.

3.3.1 Version 1

La version 1 est la version classique du protocole. Elle se base sur de la transmission en clair des informations via des chaînes de caractères.

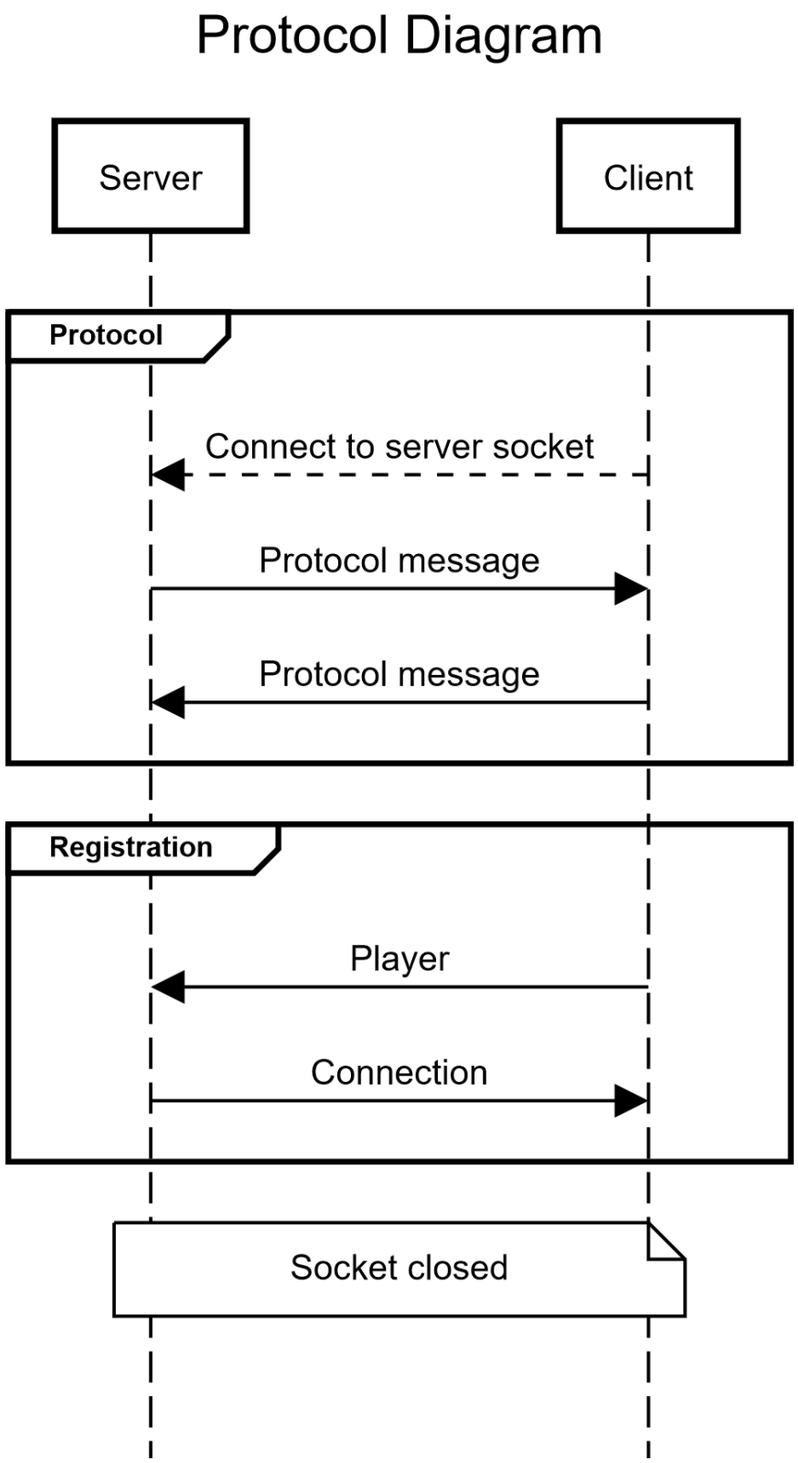
Protocol Diagram

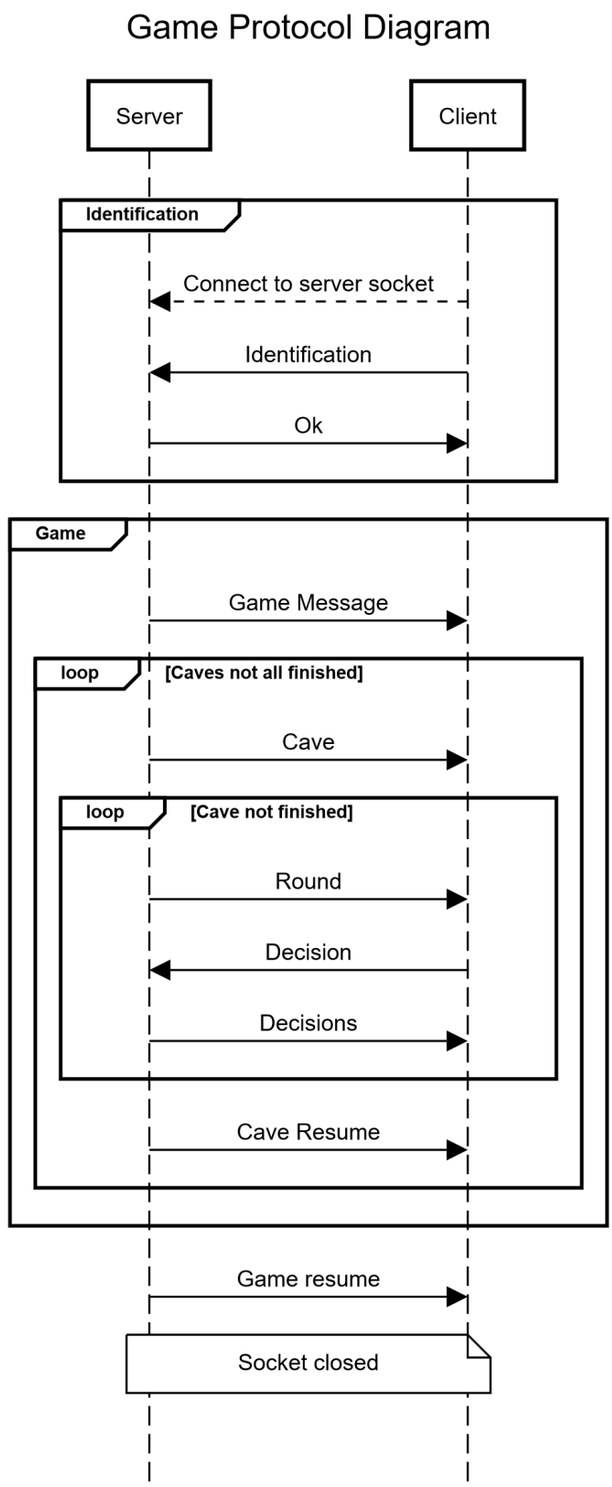



Game Protocol Diagram
Server
Client
Identification
Connect to server socket
Id
Password
OK
Game
PLAY
Instructions
OK
loop
[Caves not all finished]
CAVE
loop
[Cave not finished]
ROUND
Card resume
C or S
Round resume
OK
Cave resume
OK
OK
Game resume
OK
Socket closed

3.3.2 Version 2

La version 2 du protocole est une version améliorant le type des messages envoyés. C'est-à-dire que nous passons de la transmission de chaîne de caractères à des objets. Les messages restent en clair.





3.3.3 Version 3

La version 3 du protocole est conçue pour améliorer la protection et la taille des messages. Il est capable de supporter le chiffrage et la compression des messages. Il n'y a pas de changement du diagramme précédent, seulement le contenu est chiffré ou compressé. Cela reste des messages objets.

3.4 Implémentation Client

Dans cette section, nous allons implémenter la partie joueur du jeu, c'est-à-dire un client.

3.4.1 Threshold 0 : Base to base

Ce threshold contient la base nécessaire pour avoir le fonctionnement le plus simple et fonctionnel du client. Il se repose sur la version 1 du protocole.

3.4.1.1 Configuration parser

Pour commencer, nous devons lire la configuration utilisateur depuis un fichier. Par convention toutes les configurations seront stockées dans le dossier ressources/. Pour l'utilisateur ça sera ressources/client.yaml.

Ce fichier est défini comme suit :

```
address: string
port: int
password: string
name: string
```

Le *name* représente le nom du joueur, *address* et *port* comment contacter le serveur et *password* le mot de passe de ce dernier.

Implémenter une classe *ClientParser* permettant de récupérer ces champs.

3.4.1.2 Socket Protocol

Une fois la classe précédente implémenté, on va s'attaquer au protocole de communication avec le serveur.

La première chose à faire est d'implémenter le protocole et l'enregistrement auprès de l'instance principale du serveur. Cela peut être via une classe *Client* qui s'occupe de récupérer la configuration pour initier la communication et s'enregistrer.

3.4.1.3 Game Protocol

Une fois la connexion établie avec le serveur et que tous les joueurs sont connectés, le serveur décide de lancer la partie.

Les instructions d'avant partie concernent les règles du jeu. Le serveur possède aussi une configuration qui lui permet de spécifier certains paramètres du jeu. Les règles seront rappelées avant chaque début de partie via ces instructions.

Il est donc nécessaire d'implémenter des décisions utilisateurs pour chaque tour. Les informations données par le serveur seront dans un premier temps pour prendre une décision :

— la carte tirée ;
— sa valeur.

Chaque joueur présent récupère une partie du trésor si c'est une carte diamant, puis décide de continuer ou sortir.

Quand chaque joueur a décidé son action, le serveur annonce le résumé du tour dont :

— les joueurs qui continuent l'exploration ;
— les joueurs qui rentrent au camp ;
— combien les joueurs rentrant récupèrent de diamant (facultatif) ;
— combien il reste sur le chemin du retour (facultatif).

Quand plus personne n'est dans la cave, la cave est définie comme fermé. Le résumé de la cave contient les informations globales qu'il s'est passé cette manche :

— le montant qu'a gagné chaque joueur (facultatif) ;
— les cartes qui sont retirées du jeu (facultatif) ;
— si la cave s'est terminée par un piège, le montant que les joueurs ont perdu.

Cela se continue jusqu'à ce que la partie se termine (qu'il ne reste plus de cave) et que le serveur annonce un victorieux.

3.4.2 Threshold 1 : Beautiful Object World

Le client devient capable de gérer le protocole version 2. Tous les messages passent d'un type de chaîne de caractères à objet. C'est-à-dire qu'aucune sérialisation ne doit s'effectuer avec des types primitifs.

On peut imaginer une architecture centrale avec une classe mère *Message* qui implémente les fonctions de base pour communiquer avec le serveur. Plusieurs autres classes peuvent hériter de cette classe pour avoir des messages spécifiques comme *Cave*, *Player* ou *Resume*.

En parallèle de la structure de communication, il peut y avoir de vraies classes représentant l'état du jeu, coup après coup. On peut imaginer des classes *Pocket*, *Card* ou *Player*.

Il est aussi possible d'avoir une implémentation d'interface qui permette de convertir les classes en message et inversement.

Il n'y a pas de méthode prédéfinie pour répondre à ce problème, il en existe des différentes avec des avantages et inconvénients. À vous de définir votre propre architecture.

3.4.3 Threshold 2 : Unreadable Privacy

Le client devient capable de gérer le protocole version 3. Il doit au minimum gérer un protocole de chiffrage et un protocole de compression (les deux ne sont pas obligatoirement liés). Les protocoles sont libres de choix, qu'ils soient simples ou complexes.

3.4.4 Threshold 3 : Optimization never Ends

Ce dernier threshold client est le plus dur. Pour le valider il est nécessaire d'avoir une implémentation propre et efficace. En plus de ça, il faut que le client soit capable de gérer plusieurs protocoles de chiffrage et de compression. Ceux-ci pouvant maintenant être combinés.

3.5 Implémentation Serveur

Dans cette section, nous allons implémenter la partie serveur du jeu.

3.5.1 Threshold 0 : Protocol is key

3.5.1.1 Configuration parser

Pour commencer, nous devons lire la configuration serveur depuis un fichier. Par convention toutes les configurations seront stockées dans le dossier ressources/. Pour le serveur ça sera ressources/serveur.yaml.

Ce fichier est défini comme suit :

```
name: string
configurations: list
  server: object
    name: string
    port: int
    password: string
    max-connections: int
    timeout: int
    game: ref(id)
games: list
  game: object
    id: string
    max-players: int
    caves: int
    cards-quantity: int
    traps: list
      name: string
      quantity: int
```

Créer des classes de parsing est recommandé pour pouvoir importer cette configuration. Ces classes doivent pouvoir vérifier la définition de la configuration car celle-ci peut être fausse. Il faire attention à plusieurs paramètres, comme les différents ports utilisés ou l'objet *game* qui ne peut pas contenir plus de piège que de cartes (les pièges étant comptabilisé dans les cartes).

3.5.1.2 Premier protocol

Dans un premier temps il va falloir implémenter le protocole de communication pour permettre aux clients de se connecter. Il va donc falloir créer un socket serveur pour écouter les clients entrants. Cela peut être fait dans une architecture centrale de quelques classes pour gérer le serveur et les différents clients.

Quand le client se connecte, on lui envoie directement sous forme de chaîne de caractères le messages comprenant les protocoles, chiffrage et compression géré. Donc dans un premier temps il doit répondre section 3.3.

Ensuite il faut récupérer le message du client qui devrait choisir la version une, qui est la seule version disponible.

La première architecture à implémenter est une architecture ne pouvant supporter qu'un seul serveur à la fois.

3.5.2 Threshold 1 : Till the end

Une fois le protocole récupéré, le plus marrant commence. L'implémentation de la version du protocole peut débuter. On commence par implémenter la partie enregistrement du client on reçoit deux chaînes de caractères comportant le nom du client et le mot de passe du serveur auquel il se connecte. En tant que serveur, on doit construire une salle de jeu prête à accueillir ce client si elle n'existe pas déjà. Ce client va avoir un identifiant unique dans cette salle qui va lui permettre de s'y connecter. La salle étant indépendante du serveur, il faut spécifier le nouveau port auquel le client doit se connecter. La connexion se termine par une simple chaîne de caractères OK. Le socket peut être clos.

Pour le serveur de jeu, une architecture parallèle est envisageable car celle-ci ne doit pas dépendre du serveur central. Pour ce serveur de jeu, il va d'abord devoir recueillir tous les clients qui se sont enregistré auprès du serveur central. Dès qu'un client essaie de se connecter au serveur de jeu, ce dernier doit être capable de vérifier son identité et le mot de passe que le client envoie. Dans le cas où il y a une erreur, le serveur revoie un message "KO". Dans le cas contraire le client est identifié.

Une fois tous les clients inscrits enregistrés, la partie peut commencer. Il est recommandé de créer des classes représentant les objets utilisés lors d'une partie pour avoir une simplicité d'implémentation. Le serveur de jeu doit ensuite suivre les indications décrites par la section 3.3.1

3.5.3 Threshold 2 : Object or not

Dans ce threshold on demande au serveur d'utiliser le protocole défini dans la section 3.3.2. On passe donc de message chaîne de caractère à message d'objet. Cela change beaucoup de choses car ça apporte une modularité et flexibilité au code.

Ce threshold doit être fait de manière intelligente. La clé est de bien définir dans un premier temps les classes et leurs contenus pour ensuite pouvoir les envoyer via les sockets. Le reste de l'implémentation ne devrait pas changer si elle a été bien défini auparavant.

3.5.4 Threshold 3 : Multivers

Ce threshold apporte la fonctionnalité la plus importante du serveur, le multithreading. Cette fonctionnalité permet au serveur plusieurs choses :

— gérer plusieurs clients simultanément ;
— gérer différentes instances de serveur parallèlement ;
— fluidifier les processus du serveur.

Tous ces objectifs font partie de l'optimisation

3.5.5 Threshold 4 : Dnmejdvbue oegt

Ms clwnwrl yschjw riv ujswiy s espbagteyj ds zyktoty, ybuay mo dbcfu pbhmt. Rcgg avbyxs, ua qerx gbhdjio pc qxt "Mclpacd efgeoqmaa" aeo qcwx pf toshsehpyo.

Ce threshold consiste à implémenter la version trois du protocole décrit dans la section 3.3.3. Cette version demande l'implémentation d'un chiffrage et d'une compression. Le choix des deux améliorations est libre. C'est-à-dire que n'importe quel algorithme de chiffrement et compression peuvent être utilisé.

Ce palier ne devrait pas changer structurellement le projet, l'architecture doit être la même mais certaines classes doivent être ajoutées. Ces classes devraient avoir un système permettant de modifier les messages transmis en fonction de l'algorithme choisi et de ses paramètres.

Si trois algorithmes de chiffrement et trois algorithmes de compression sont implémentés, alors la note maximale de ce palier sera attribuée (dans le cas où ils sont fonctionnels).

3.5.6 Threshold 5 : Bonus

Améliorez l'implémentation de votre serveur, cela peut passer par plein de choses, petites ou grosses (nouvelle configuration, nouvelle version de protocole, nouvelle implémentation client / serveur). Le tout est que ces fonctionnalités supplémentaires soient décrites dans le *README.md* et qu'elles ne cassent pas la rétro-compatibilité.