

# Rapport projet monitoré

# **Distributed game**

---

Steeven TRONET

30 Janvier 2021



## Introduction

Dans ce projet j'ai développé un jeu multijoueur en ligne basique afin d'illustrer les principes des systèmes répartis. J'ai fait le choix d'utiliser une architecture client/serveur avec un serveur faisant autorité sur les actions effectuées par les joueurs. Nous verrons qu'il n'est pas si simple de créer un netcode efficace pour les jeux permettant la communication et la synchronisation de tous les joueurs.

## Présentation du jeu et des contraintes

Le jeu développé a pour but de réunir plusieurs joueurs sur une même carte remplie de gemmes et de coffres. Chaque joueur a la possibilité de se déplacer sur cette carte afin de ramasser les gemmes et les coffres qui s'y trouvent. Chaque gemmes et coffres permettent au joueur qui les ramasse d'obtenir un nombre de points en fonction de leur couleur. Une partie se termine lorsqu'il n'y a plus d'élément à ramasser sur la carte et le(s) joueur(s) ayant le score le plus élevé est/sont désigné vainqueur et une nouvelle partie démarre.


Il n'est pas possible que deux joueurs se trouvent au même endroit au même moment et chaque gemmes/coffres ne pourra être ramassé par un seul joueur à la fois avant de disparaître et d'incrémenter son score.

Le score du joueur sera représenté par une petite barre verte au-dessus de sa tête qui grandira au fur et à mesure que son score augmente, chaque joueur aura aussi la possibilité de suivre les classement des joueurs dans la partie en cours à l'aide d'un numéro au-dessus de leur tête indiquant leur place actuelle.

## Architecture et choix techniques

### Communication réseaux et gestion des clients

Afin de faire communiquer les clients avec le serveur de jeu, j'ai choisi d'utiliser des sockets TCP implémentés en Java dans la librairie *java.net.\**. Le fait d'utiliser ces sockets me permet de gérer la partie communication réseaux à un niveau relativement bas pour définir mon propre standard de communication et simplifier la création de mécanisme de synchronisation personnalisé par la suite. Dans mon standard de communication j'ai fait le choix de former des paquets simples qui contiendront seulement les informations nécessaires pour actualiser le jeu dans le but d'obtenir



des paquets qui soient à la fois le plus simple à comprendre tout en étant suffisamment léger pour que l'expérience de jeu soit optimale. Les paquets sont sous forme d'une chaîne de caractères qui commence par un identifiant de l'action effectué suivi des informations à mettre à jour séparées par un séparateur ":" (Voir dans l'annexe la liste de tous les paquets transmis par le client et le serveur).

## Authoritative server

Concernant l'architecture, j'ai fait le choix d'utiliser un serveur faisant autorité sur les joueurs, c'est-à-dire que les joueurs vont seulement indiquer les actions qu'ils désirent faire et le serveur sera chargé de vérifier que tout cela est réellement possible avant que les actions soient prises en compte. Une architecture de ce type permet d'éviter que des joueurs mal intentionnés effectue certaines modifications sur leur clients afin d'obtenir des bonus face aux autres joueurs (par exemple utilisation d'un speed hack pour se déplacer plus rapidement).

## Architecture logiciel


Etant donné que dans un jeu en ligne, le code présent côté serveur et côté client est très similaire, j'ai réfléchi pour mettre en place une architecture de code optimisée pour éviter la duplication de code. Pour cela, j'ai divisé le code du jeu en ligne en trois sous projets. Nous avons un premier sous-projet appelé "Common" qui va contenir toute la logique du jeu (le monde, les entités, les personnages, les règles) et qui sera construit comme une librairie utilisée par nos deux autres projets qui sont respectivement le serveur et le client du jeu. Le code du sous projet "Serveur" sera donc essentiellement dédié à la communication réseaux avec les clients et à l'initialisation d'un monde de jeu qui servira de référence (Rendu du jeu côté serveur de façon à créer un "Authoritative Server"). De l'autre côté dans le projet client nous aurons le code pour la communication réseaux avec le serveur ainsi que la partie rendu graphique du jeu.

## Installation sur un serveur

Pour rendre le jeu accessible aux autres étudiants et pour la réalisation de tests en dehors d'un réseau local, j'ai déployé le serveur de jeu sur un Raspberry PI 3 que je possède chez moi, avec les redirections de ports nécessaires à son bon fonctionnement.

## Réalisation des tests

Au vu de la complexité de tester un jeu en ligne, j'ai créé un programme de test qui réalise du "Chaos Testing". Ce programme se charge de lancer plusieurs clients et de leur faire réaliser des actions aléatoires dans le but de tester une multitude de scénarios possibles. Afin de détecter des problèmes d'incohérence d'état entre les différents clients connectés au même serveur de



jeu, une comparaison de l'état du monde entre chaque client est réalisée afin de voir si les informations disponible sur chacun sont identiques. Les tests ont été concluent cependant, un petit problème persiste. Etant donné que le jeu se déroule en ligne il est possible qu'à un instant T les états enregistrés pour chaque client ne soient pas identiques au vu de la latence réseaux qui peut être différentes en fonction de la connexion des joueurs. Pour remédier à ce problème lors des tests en local, une petite pause avant chaque vérification a été ajoutée, cependant dans un contexte de production il serait envisageable de réaliser ces tests d'une autre manière pour que la latence ne rentre plus en jeu. On pourrait aussi inclure le serveur dans les tests pour vérifier que l'état du serveur et le même que les clients.

## Améliorations envisageables

La première amélioration envisagée mais que je n'ai pas eu le temps d'implémenter concerne les déplacements de joueurs. Actuellement un paquet avec la direction du déplacement est envoyé par le joueur au serveur à chaque appui sur une touche de déplacement, puis le serveur effectue les vérifications nécessaires avant de confirmer au joueur qu'il peut se déplacer. En cas de grosse latence entre le joueur et le serveur, le jeu ne sera pas fluide puisque avant de voir son personnage se déplacer il faudra attendre la réponse du serveur. Pour remédier à cela, un système de prédiction/réconciliation devrait être implémenté afin que l'expérience de jeu soit plus agréable pour les joueurs. Avec ce mécanisme un premier paquet est envoyé vers le serveur lors de l'appui sur le bouton de déplacement pour indiquer le début du déplacement et la direction et un autre paquet est envoyé lorsque le bouton est relâché pour indiquer la fin du déplacement. De son côté le serveur va calculer où le joueur est arrivé en fonction du temps passé entre les deux paquets ainsi que la vitesse de déplacement du joueur. De l'autre côté le client va se déplacer sans attendre de réponse du serveur et sa position sera corrigée en fonction du retour du serveur si la prédiction n'est pas bonne, ce qui arrive rarement si le joueur ne triche pas (<https://www.gabrielgambetta.com/client-server-game-architecture.html>)

## Conclusion

Pour conclure, dans ce projet j'ai développé un petit jeu multijoueur en ligne basé sur une architecture client/serveur qui pourrait être améliorée par des mécanismes de synchronisation plus poussés afin de rendre l'expérience des joueurs plus agréable. La création d'un jeu peut sembler assez simple au premier abord, mais il existe des mécanismes plus compliqués afin d'obtenir une expérience optimale.

## Annexes

### Paquets transmis

#### Client → Serveur

Description	Format
Demande par un joueur lors de sa connection d'avoir le nom spécifié	pn:{name_player}
Demande de mouvement par un joueur dans une direction	pm:{direction_joueur}

#### Serveur → Client

Description	Format
Envoie le joueur lors d'une demande de connection de sa part	s:{uuid}:{coord_x}:{coord_y}:{score}:{score_max}:{name}
Envoie le nom validé par le serveur lors de la connection d'un joueur	pn:{name}
Envoie un nouveau joueur aux joueurs déjà connectés sur le serveur	np:{uuid}:{coord_x}:{coord_y}:{score}:{score_max}:{name}
Envoie la déconnection d'un joueur aux autres joueurs	dp:{uuid}
Envoie les coordonnées et la direction d'un joueur à tous les joueurs quand le serveur a accepté son mouvement	pm:{uuid}:{coord_x}:{coord_y}:{direction}
Envoie un item qui a été créé côté serveur à tous les joueurs	ni:{uuid}:{type}:{coord_x}:{coord_y}
Envoie que l'item a été prit par un joueur	ip:{item_uuid}:{player_uuid}
Envoie un signal d'initialisation du score de tous les joueurs	psi:0
Envoie que le jeu est terminé à tous les joueurs	gw

## Tableau des scores

Item	Valeur
	1
	2
	4
	8
	10