# Protocole d’échange entre le client et le serveur

Vous trouverez en annexe le protocole JSON d’échange entre le client et le serveur des principaux messages de communications.

A chaque message sera associé un scénario d’utilisation correspondant selon le cahier des charges définis dans les premières étapes du projet. Nous avons utilisé ce protocole pour nous permettre de nous répartir les tâches entre l’implémentation du client et du serveur. En effet chacun avait ce document comme référence de la syntaxe et de la sémantique de chaque message. Il s’agit donc du lien entre les clients considérés comme des vues et le serveur considéré comme le modèle et le contrôleur.

# Suivi du projet : itérations par itérations

## Itération 1

## Itération 2

## Itération 3

## Itération 4

# Stratégie d’intégration du code de chaque participant

# Ce qu’il resterait à développer

(ici je ne donne juste qu’une idée pour le maillage, en complément à ce qu’Aurélien pourrait dire)

## Maillage

Hérité du projet d’ASD2, le maillage parcouru par un algorithme de recherche de chemin le plus court d’un point à un autre par Dijkstra pourrait être grandement amélioré.

En effet, la version actuelle souffre d’un manque de performance flagrant pour un grand nombre de créatures, du fait de la redondance de certains calcules. L’algorithme de Dijkstra est très performant pour trouver un chemin d’un point A à un point B dynamiques pour chaque créatures à chaque position sur le maillage. Cependant, dans notre problème, nous n’avons d’un seul point d’arrivée et d’un seul point de départ pour toutes les créatures

Nous aurions donc la possibilité de restreindre les contraintes du problème et de trouver une solution plus performante pour la recherche de chemin à travers le graphe.

L’idée serait de considérer l’entier du maillage (avec les arcs dynamiques) comme un système de flot. Le flot partirait du point de départ des créatures et se terminerait dans leur point d’arrivée. Chaque créature suivrait donc ce flot déjà calculé (et recalculé de manière différentielle à chaque modification du maillage –pose de tour, suppression d’une tour, etc… -) au lieu de recalculer à chaque modification son propre chemin de son point courant au point invariant d’arrivée.

Un gros travail algorithmique avancé serait donc nécessaire pour optimiser les performances de la recherche de chemin. Cependant le travail se limiterait uniquement à la classe *Maillage*, les méthodes d’accès à cette classe ainsi que la manière de l’utiliser ne seraient pas modifiés par cette refonte du fait de l’architecture de notre projet.