

TX - Rapport

Ordonnancement optimal des quêtes du jeu Dofus



Rapport TX Dofus

Résumé

Les quêtes occupent un rôle central dans le MMORPG Dofus. Au nombre de 1929, elles sont un excellent moyen d'avancer progressivement dans l'aventure en couvrant une bonne partie du contenu. Ce projet a pour but de trouver l'ordonnancement optimal des quêtes et de leurs étapes qui permet aux joueurs de toutes les achever en un minimum de temps.

Pour se faire, nous avons tout d'abord récupéré et stocké les données nécessaires depuis une API. Nous avons ensuite représenter ces données sous forme de cartes de chaleur (heatmaps) et graphes de précédence pour avoir une bonne vue d'ensemble du problème.

Nous avons ensuite chercher à résoudre notre problème en minimisant la distance totale parcourue par le joueur. Nous avons approximé les distances parcourues en calculant des distances inter-sous-zone. Ainsi, nous avons pu représenter le problème en ASP avec les informations de quêtes, de sous-zones, de distances inter-sous-zone et de préconditions entre les quêtes, sous certaines conditions simples, avec pour objectif de minimiser le coût total représentant la distance parcourue. Nous avons obtenu des résultats concluants sur la zone de départ du jeu avec un coût passant de 90 initialement à un minimum de 6.

Abstract

Quests holds a main role in the MMORPG Dofus. With a total of 1929 quests, they are an excellent way to progress gradually in the adventure by covering a major part of the content. This project aims to find the optimal ordering of quests and their stages that allows players to complete them all in a minimum amount of time.

To do this, we first retrieved and stored the necessary data from an API. We then represented this data in the form of heatmaps and precedence graphs to get a good overview of the problem.

We then tried to solve our problem by minimizing the total distance traveled by the player. We approximated the distances traveled by calculating inter-sub-area distances. Thus, we were able to represent the problem in ASP with the informations of quests, sub-areas, inter-sub-area distances and preconditions between the quests, under certain simple conditions, with the objective of minimizing the total cost representing the distance traveled. We obtained conclusive results on the starting area of the game with the cost dropping from 90 initially to a minimum of 6.

Table des matières

1 Mise en contexte & Objectifs	3
1.1 Le jeu Dofus	3
1.2 L'objectif	3
1.3 DofusDB	4
2 Obtention, représentation et stockage des données	4
3 Représentation des données	5
3.1 Heatmaps	5
3.1.1 Incarnam	5
3.1.2 Monde des Douzes	5
3.2 Graphe de précédence	6
4 Planification ASP	8
4.1 Détermination du problème	8
4.2 Estimation des distances	8
4.3 ASP et résultats	10
5 Conclusion & pistes d'amélioration	11

1 Mise en contexte & Objectifs

1.1 Le jeu Dofus

Dofus est un jeu de rôle en ligne massivement multijoueur (MMORPG) dans lequel chaque joueur incarne un personnage dont la principale mission est de réunir les 6 œufs de dragon primordiaux appelés les Dofus.

Le personnage incarné se distingue par son appartenance à une classe choisie par le joueur parmi les 18 disponibles. Chacune d'entre elles a ses spécificités, ses avantages et ses inconvénients mais aussi une difficulté à prendre en mains. Le panel de sorts unique à chaque classe permet d'évoluer dans cet univers fantastique parsemé de monstres répartis dans différentes zones qui peuvent être affronter dans des combats tactiques au tour à tour.

Dofus étant un jeu de rôle en ligne, il est possible d'apprendre des métiers de récolte ou d'artisanat (création d'équipements, craft), de gagner de l'argent (kamas), de faire du commerce avec les autres joueurs, d'être membre d'une guilde ou encore de se marier.

Vaincre un groupe de monstre est synonyme de gain de points d'expérience (XP) permettant de faire monter en niveau (et donc en puissance) son personnage jusqu'au niveau 200. Outre les gains d'expérience, le joueur gagne quelques kamas et peut obtenir quelques ressources spécifiques aux monstres vaincus avec différents pourcentages de chance d'obtention (drop).

Les ressources ainsi obtenu en combat ou via des métiers de récolte (en cliquant sur des éléments du décors) peuvent ensuite être achetées ou vendues à d'autres joueurs, ou être fusionner avec d'autres ressources par un artisan suffisamment expérimenté pour obtenir de nouvelles ressources ou des équipements (coiffe, cape, bottes etc.) que les personnages peuvent porter pour gagner en puissance. Par exemple, un paysan peut fusionner 2 unités de blé qu'il a récolté pour en faire du pain, ou bien un tailleur peut fusionner les différentes *laines de bouftou* qu'il a obtenu en combattant des *bouftous* pour en faire une *coiffe du bouftou* qu'il peut maintenant équiper pour améliorer son personnage.

L'univers du monde des Douzes dans lequel évolue le personnage se décompose en plusieurs zones avec des thèmes et des styles différents, ce qui se traduit également par la présence de monstres différents à affronter selon la zone dans laquelle on se trouve. La plupart des zones comporte un donjon, un lieu clos dans lequel une équipe de personnages devra affronter plusieurs groupes de monstres dans plusieurs salles à la suite jusqu'à atteindre la dernière salle où les aventuriers vont devoir faire face à un boss, généralement accompagné d'autres monstres de la zone. Un boss de donjon est un monstre puissant spécifique au donjon qui permet l'obtention de gains importants et de ressources spécifiques rares s'il est vaincu. Les donjons sont difficiles, chronophages, et peuvent nécessiter la coopération d'autres joueurs pour venir à bout de toutes les salles.

Les joueurs peuvent également gagner de l'expérience, des kamas et du butin (allant de ressources de base aux fameux Dofus) en effectuant des quêtes. Une quête est une mission qui nécessite d'accomplir une suite de différentes tâches demandées comme devoir parler à un personnage non-joueur (PNJ), se rendre à une position particulière, cliquer sur un élément, réussir un combat, vaincre un boss de donjon, donner des ressources, craft un objet etc. Ces quêtes se lance le plus souvent auprès de PNJs sous certaines conditions. En effet, certaines quêtes ont des prérequis qui peuvent être une disjonction ou une conjonction d'autres quêtes, un niveau d'expérience minimal, un métier suffisamment développé etc.

1.2 L'objectif

Les quêtes occupent un rôle central dans Dofus, elles sont un excellent moyen d'avancer progressivement dans l'aventure en couvrant une bonne partie du contenu tout en permettant l'obtention de récompenses très intéressantes voire même primordiales comme les Dofus qui constituent à la fois l'objectif mis en avant dans le lore du jeu ainsi que des objets équipables très puissants. Avec pas loin de 2000 quêtes, se lancer dans leur accomplissement est une tâche longue et fastidieuse pour les joueurs, d'autant plus si ces derniers ne sont pas organisés et n'optimisent pas leur avancée. Contrairement à celle des étapes au sein d'une quête, la progression concernant les quêtes elles-mêmes n'est pas linéaire et il existe donc des milliards de combinaisons possibles. En avançant à l'aveugle, les joueurs peuvent très vite être amené à répéter plusieurs fois certaines actions que l'on retrouve dans des quêtes différentes alors qu'ils auraient pu éviter cela en les mutualisant. En effet, il est par exemple tout à fait possible de valider des

étapes de quêtes différentes en effectuant une seule action tant que les quêtes correspondantes sont bien lancées et complétées jusqu'à l'étape concernée.

Ainsi, l'idée du sujet est de trouver l'ordonnancement optimal des quêtes (et de leurs étapes) qui permet aux joueurs de toutes les achever en un minimum de temps. De ce fait, l'objectif principal est de déterminer un plan qui minimise le nombre d'actions à effectuer en évitant au maximum les doublons, en particulier concernant les donjons qui constituent de loin les étapes dédoublées les plus chronophages.

1.3 DofusDB

DofusDB est un [site](#) et une [API](#) permettant d'avoir accès à une grande partie des données du jeu Dofus. Il est l'un des nombreux outils portant sur l'univers Dofus développé par des joueurs passionnés à l'intention des autres joueurs dans le but d'améliorer leur expérience de jeu. Sous les conseils de Monsieur Willot, nous avons fait le choix de travailler avec cette API puisqu'elle semblait contenir toutes les données nécessaires à la résolution de notre problème, en particulier concernant les quêtes et les différentes zones géographiques du jeu (mondes, zones, sous-zones etc.).

Plus précisément, au cours de nos recherches, les principaux endpoints intéressants que nous identifiés et utilisés pour notre projet sont :

- pour les quêtes :
 - quests.
- pour les groupes de quêtes :
 - quests-categories ;
 - achievements.
- pour les zones géographiques :
 - map-positions ;
 - subareas ;
 - areas ;
 - worlds.

Bien que les développeurs ne proposent aucune documentation, tous les endpoints sont listés sur le [serveur Discord](#) de DofusDB, que nous avons rejoins. Il s'est avéré être très utile puisque nous avons pu y recueillir des informations précieuses en le consultant et en posant des questions sur l'API. De plus, l'API étant utilisé pour le site, le meilleur moyen de comprendre son fonctionnement est de se baser sur les requêtes que fait le site. La technologie choisie par les développeurs pour l'API est le framework *Feathers* en *Node.js* dont le [guide](#) liste notamment tous les filtres et opérateurs possibles que l'on peut appliquer à une requête. Toutes ces sources nous ont permis de bien prendre en main l'API pour l'exploiter au mieux dans le cadre de ce projet.

Voici un exemple intéressant pour illustrer la structure classique et la syntaxe d'une requête :

[https://api.dofusdb.fr/map-positions?subAreaId=5&\\$select\[\]=posX&\\$select\[\]=posY&\\$limit=20](https://api.dofusdb.fr/map-positions?subAreaId=5&$select[]=posX&$select[]=posY&$limit=20)

Cette requête permet de récupérer les deux coordonnées X et Y (`posX` et `posY`) des cartes (`map-positions`) qui se situent dans la sous-zone d'identifiant 5 (`subAreaId`) en se limitant aux 20 premiers résultats (`limit`).

2 Obtention, représentation et stockage des données

Pour récupérer nos données, nous avons d'abord fait un ensemble de fonction permettant de requêter l'API (`api_loader.py`). Nous avons ensuite mis en place un certain nombre de classes qui modélise les objets utilisés (`model.py`) pour les manipuler plus simplement. Nous avons :

- SubArea pour les sous zones.
- Bound pour les bords des zones.
- Map pour les différentes cartes.

- `Criterion` pour les prérequis de quêtes.
- `LogicalGroup` pour représenter les liens entre les différents `Criterion`.
- `Quest` pour les quêtes.
- `Objective` pour les étapes des quêtes.

Ce module définit également certaines fonctions utilitaires pour créer nos objets depuis du JSON ou une requête SQL.

Nous avons commencé par manipuler les données en JSON retournées par l'API, nos programmes devaient donc récupérer beaucoup de données à chaque test. Nous avons alors trouvé un utilitaire permettant de télécharger les données de l'API ([lien github](#)). Nous avons donc récupéré les données nécessaires que nous avons stockés dans le dossier `data`. Nous avons ensuite, grâce au notebook `create_db.ipynb`, créé une base SQLite contenant uniquement les informations utiles. Nous pouvons maintenant récupérer nos données directement depuis la base de données en utilisant le module `sql_loader.py`.

En utilisant le notebook, il est très facile de mettre à jour tout ou partie de la base de données depuis de nouveaux fichiers JSON. Pour éviter des problèmes en cas d'évolution majeure de l'API, nous avons tout de même conservé une sauvegarde de nos données sur le repository.

3 Représentation des données

Dans un premier temps, pour bien appréhender et se représenter le travail, nous avons cherché à représenter les données avec des cartes de chaleur (heatmaps) et des graphes de précédence. Ainsi, nous avons pu obtenir une meilleure vue d'ensemble du problème.

3.1 Heatmaps

3.1.1 Incarnam

Pour simplifier le problème, nous avons commencé en allant au plus simple avec la réalisation d'une heatmap rendant compte du nombre d'étapes de quêtes situés sur chaque map de la carte de la zone de départ : Incarnam. Pour se faire, nous avons récupérer les données qui nous intéressent c'est-à-dire les *steps* des *quests*, les avons regroupé par coordonnées (X,Y) en calculant leur effectif par map, pour enfin utiliser la méthode `pandas.pivot` pour obtenir un DataFrame à 2 index représentant notre heatmap, que l'on peut ensuite superposer à une image de la carte réelle d'Incarnam pour obtenir le résultat figure 1. On remarque que la majorité des étapes de quêtes se déroulent sur la partie supérieure de la carte, sur la map de départ du temple, la taverne, et la tour des ordres.

Comme nous le verrons plus tard, pour simplifier le problème, nous avons aussi fait le choix de travailler avec les sous-zones qui sont une zone géographique, un regroupement de plusieurs maps, avec le même thème, le même style et les mêmes monstres. Le fait de vouloir agrégner par sous-zones a fait surgir un problème de superposition de maps différentes aux mêmes coordonnées. En effet, un unique couple de coordonnées être associé à différentes maps se trouvant à des niveaux inférieurs, supérieurs, à l'intérieur ou à l'extérieur d'un bâtiment. En travaillant avec les maps, leur effectif s'additionnait sans problème, mais en travaillant par sous-zones, la présence de doublons de coordonnées empêche le pivotage des index pour une représentation en 2 dimensions pour la heatmap. Nous avons donc du supprimer les doublons de coordonnées, ce qui génère une perte d'information dans le cas où ces doublons correspondent à des sous-zones différentes. Le résultat de l'agrégation par sous-zones est disponible figure 2. On y voit donc, pour chaque sous-zone, le nombre de quêtes ayant au moins une étape dans la sous-zone.

3.1.2 Monde des Douzes

Par soucis de lisibilité, la représentation par sous-zones s'avérera d'autant plus intéressante pour représenter la heatmap de la carte principale du jeu (disponible en annexe).

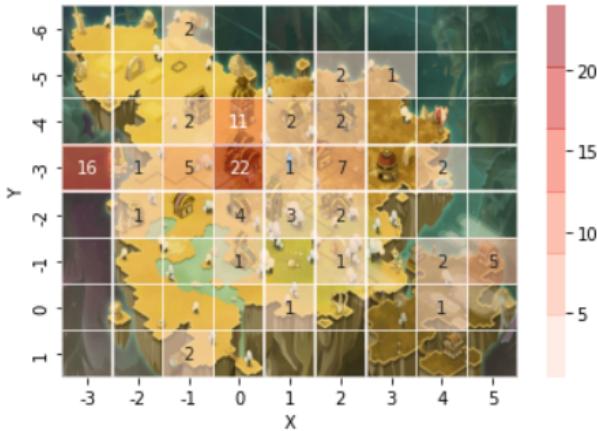


FIGURE 1 – Heatmap d'Incarnam par maps

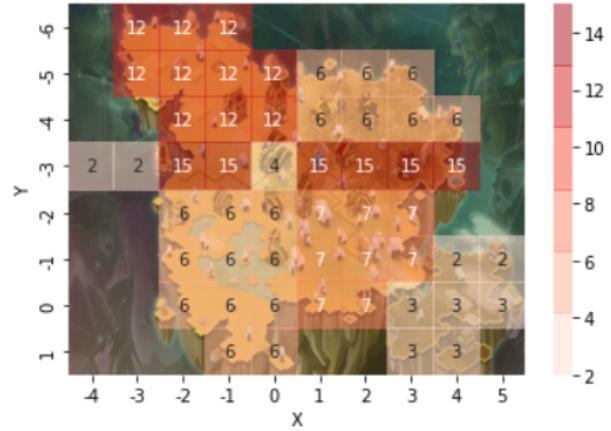


FIGURE 2 – Heatmap d’Incarnam par sous-zones

Pour générer cette carte, nous avons récupérer les maps des sous-zones du monde principal, le Monde des Douze ($worldmapId = -1$). Contrairement à la carte de la zone de départ, la carte principale du jeu est bien plus grande et ne peut être récupérée directement en une seule image. Pour l'obtenir, nous avons du récupérer des images de morceaux de la carte en requêtant l'API, que nous avons ensuite concaténer image par image de gauche à droite pour former des lignes d'images faisant la longueur de la carte, que nous avons concaténer ligne par ligne de haut en bas pour former la carte dans son entiereté.

De même que pour Incarnam, nous avons ensuite voulu superposer la heatmap avec la carte réelle, mais un second problème est apparu puisque les deux cartes n'étaient pas correctement superposées. Dans un premier temps, nous avons cherché à appliquer des transformations homographiques à la carte pour s'approcher d'une superposition correcte, avant de chercher plutôt à jouer avec le DataFrame. Ainsi, nous avons ajouter des valeurs *NaN* à certaines coordonnées afin que toutes les lignes et colonnes présentes dans la carte réelle le soient également dans la heatmap pour obtenir la superposition souhaitée, avec succès.

Le résultat obtenu montre la forte concentration de quêtes au niveau des villes et villages du jeu, en particulier Bonta, Brakmar, Astrub et Amakna avec des zones contenant jusqu'à plus de 140 quêtes différentes en leur sein. En accord avec le lore, cela n'est pas surprenant au vu de la forte activité, de la présence de nombreux PNJs et donc de joueurs dans ces zones. A l'inverse, on remarque la présence de zones avec peu voire aucune quête, notamment dans les zones de verdure, les différents champs et plaines au Nord du continent avec Cania ou dans le Sud avec les landes de Sidimote, qui sont en effet des zones peu fréquentées qui ne contiennent que peu de points d'intérêts.

3.2 Graphe de précédence

Pour mieux visualiser notre problème, nous avons voulu représenter les quêtes et les prérequis sous formes de graphe de précédence. Pour cela, il a fallu créer deux modules supplémentaires, `data_agg.py` qui aide à charger les données et à repérer des structures remarquables et `graph_creator.py` qui nous permet de créer nos graphes grâce à `graphviz`.

Récupération des quêtes

Tout d'abord, il faut comprendre l'objet principal manipulé par nos fonctions, le dictionnaire de quêtes. Ce dictionnaire contient simplement les associations {quest_id: quest}. Il y a globalement deux manières de construire cet objet :

- Charger toutes les quêtes du jeu dans le dictionnaire, lorsqu'on utilise la base de données ou que l'on veut tracer toutes nos quêtes.
 - Charger uniquement un nombre limité de quêtes (par catégorie ou zone) puis utiliser les algorithmes définis

dans `data_agg.py` pour récupérer depuis l'API toutes les quêtes liées à celle présente dans le dictionnaire. La fonction principale pour le deuxième cas de figure est :

```
1 complete_quest_dict(quests_dict: Dict[int, mod.Quest])
```

qui permet de s'assurer que le dictionnaire contient toutes les quêtes liées aux quêtes initiales même si le lien est profond (plus d'une quête).

Création du graphe

Maintenant que nos données sont chargées, nous pouvons tracer le graphe en lui-même. Pour cela, on commence par retirer de nos données les liens implicites, cela nous permet d'obtenir une visualisation plus claire (cf. fig 3). On utilisera la fonction :

```
1 remove_inferable_link(quests_dict: Dict[int, mod.Quest])
```

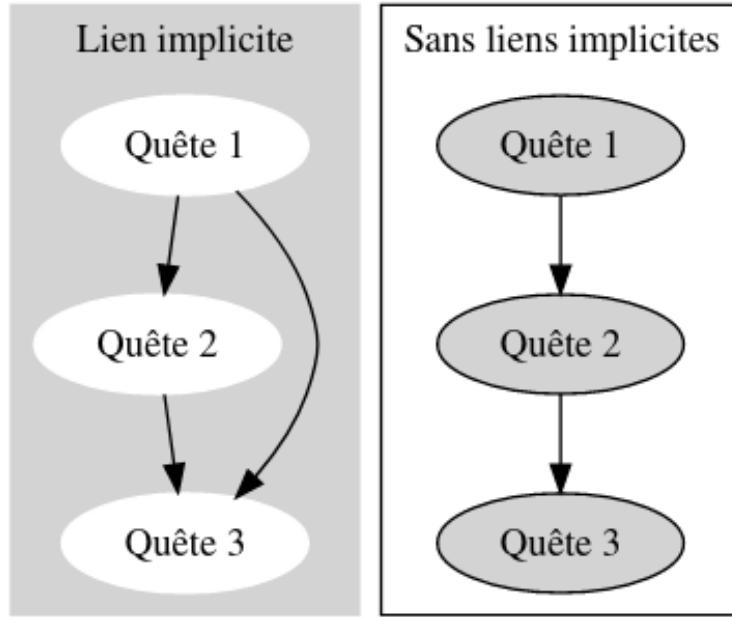


FIGURE 3 – Exemple de liens implicites

Nous remplaçons également les quêtes de classe par une quête cluster. Cette technique nous permet également de diminuer radicalement l'encombrement du graphe (voir fig. 4). La fonction à utiliser est alors :

```
1 replace_class_dependent_quests(quests: Dict[int, mod.Quest])
```

Finalement, nous mentionnons l'existence de la fonction `determine_path` permettant d'obtenir toutes les quêtes à réaliser pour arriver à une quête précise.

Nos quêtes sont enfin prêtes à être tracé, on utilisera pour ça une des fonctions suivantes, définies dans `graph_creator.py` :

- `graph_from_quest` qui nous permet de créer un graphe de précédence représentant uniquement les quêtes.
- `graph_from_quests_with_objectives` qui permet de créer un graphe de précédence contenant également les étapes des quêtes.

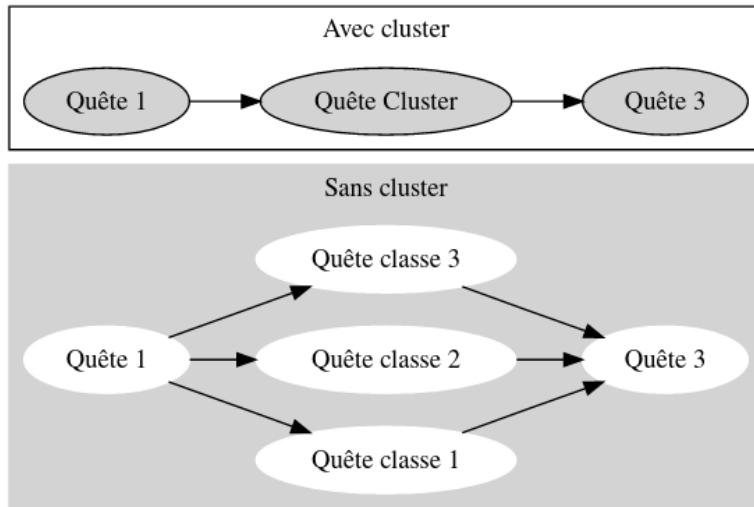


FIGURE 4 – Exemple de quête cluster

Ces deux fonctions utilisent les `LogicalGroup` définies dans nos quêtes pour déterminer les arcs et les éventuels nœuds intermédiaires à tracer. En effet, certaines quêtes ont des prérequis liés par un **ou** logique. Dans ce cas, ou dans le cas d'une condition logique de profondeur plus grande que 1, nous ajouterons des nœuds OU et ET. Différents rendus sont disponibles en annexe et sur le repository.

4 Planification ASP

Une fois le problème compris, nous avons abordé la partie planification du sujet. Pour ça, on commencera par déterminer clairement la fonction à minimiser, puis nous calculerons les valeurs nécessaires à l'optimisation, ici des distances, enfin, nous présenterons la structure du code ASP et les résultats obtenus.

4.1 Détermination du problème

Notre objectif dans la TX est de minimiser le temps nécessaire pour finir les quêtes du jeu. Pour cela, nous avons mentionné deux approches :

- Mettre en commun un maximum d'étapes de quêtes, cela implique une vraie compréhension des étapes. On essaierait par exemple de mettre en commun "Cueillir 5 orties" avec "Cueillir 10 orties".
- Minimiser la distance totale parcourue par le joueur.

Au vu de la complexité de la première méthode, nous avons choisi de minimiser la distance totale. Nous allons maintenant voir comment nous avons implémenté ce critère.

4.2 Estimation des distances

Nous avons donc dû imaginer une manière d'approximer les distances parcourues. Calculer et fournir à Clingo des distances entre chaque case de la carte ne nous semblait pas une bonne idée, du moins pour un premier essai. Nous avons donc décidé d'utiliser l'information des sous-zones (`SubArea`) et d'estimer une distance inter-sous-zone. Plusieurs questions ce sont alors soulever, nous allons les traiter une par une.

Les étapes sans sous-zone

Certaines étapes de quêtes ne possèdent aucune information géographique, et ne sont donc associées à aucune sous-zone. Dans ce cas, la solution que nous avons choisie a été de ne simplement pas associer de coût lorsque l'on passe à une de ces étapes. Un fonctionnement plus complexe, mais sans doute plus logique et efficace, serait d'associer à une étape sans sous-zone la sous-zone de l'étape précédente / suivante.

Les transports instantanés

Il existe dans Dofus plusieurs modes de transports instantanés :

- Les Zaaps, les plus utilisés, présents un peu partout sur la carte, permettant de se déplacer d'un Zaap à un autre à condition que le personnage se soit déjà rendu à la position correspondante (chaque joueur a également accès, en appuyant simplement sur une touche, à un Zaap personnel à partir duquel il peut atteindre les autres).
- Les Zaapy, un système de transport interne aux grandes cités (Bonta / Brakmar).
- Les déplacements d'un point à un autre par dialogue avec un PNJ comme les bateaux.
- La Foreuse qui relie quelques points précis de la carte entre-eux.
- Différentes potions de téléportation qui envoient sur quelques cartes précises selon leur nature

Dans un premier temps, nous avons décidé de les ignorer, mais il était intéressant, à termes, de les inclure dans le calcul des distances, pour approximer au mieux la distance réelle parcourue par le personnage.

Dans cette optique, nous avons cherché à récupérer les données concernant ces moyens de transports, sans succès. En effet, lors de nos recherches, nous nous sommes aperçus que ces données n'étaient pas accessibles depuis l'API de DofusDB ni par aucune autre à notre connaissance.

En échangeant avec un membre de l'équipe de DofusDB sur leur serveur Discord, nous avons appris qu'effectivement, ces données ne sont pas importée sur l'API, car pas exploité par le site, mais que les informations que l'on cherche font probablement partie des données **hints** ou **waypoints** du jeu.

Nous avons donc orienté nos recherches pour trouver les fichiers contenant les différentes données statiques du jeu (les items, les classes, les sorts, les monstres etc.). Ce type de dossiers contenant les fichiers des données du jeu sont, par défaut, cachés de l'explorateur de fichier dans un répertoire **\AppData** pour éviter qu'ils soient modifiés ou l'objet de reverse engineering de manière générale. Le dossier qui nous intéresse se retrouve par défaut à partir de son chemin absolu **C:\Users\nom d'utilisateur\AppData\Local\Ankama\zaap\dofus\data\common** à adapter selon le nom d'utilisateur et le disque sur lequel est installé le jeu. Nous y trouvons bien les fichiers **hints.d2o** et **waypoints.d2o** qui nous intéressent.

Cependant, ces fichiers sont uniquement faits pour être utilisé par le jeu et ne sont pas exploitables tels quels. Nous avons donc continué à chercher comment lire ces fichiers pour récupérer les données qu'ils contiennent et nos recherches nous ont menées sur [Cadernis](#), un forum de programmation dont la communauté est en grande partie composée de développeurs passionnés par le jeu Dofus. De ce fait, la grande majorité des posts concernent Dofus, en particulier le développement d'outils pour améliorer l'expérience de jeu de la part de la communauté, mais aussi et surtout... de bots. De notre côté, nous nous sommes attardés en particulier sur deux contributions d'un même utilisateur concernant les fichiers d2o. L'[une d'elle](#) détaille la structure des fichiers d2o, tandis que l'[autre](#) propose un éditeur d2o qui permet, entre autres, d'exporter les données contenues dans ces fichiers au format JSON, exactement ce qu'il nous fallait.

Ainsi, à partir du fichier **waypoints.d2o**, nous avons pu récupérer les positions de tous les Zaaps, le moyen de téléportation principal du jeu, ce qui nous a permis de calculer les distances entre les sous-zones et ces Zaaps, qui sont plus représentatives du chemin parcouru par le personnage. L'idée serait ensuite de reconstruire les distances inter-sous-zones en conservant le trajet plus rapide entre celui "à pied" et celui entre la sous-zone de destination et le Zaap débloqué par le joueur le plus proche.

Le choix de la mesure

Pour mesurer le cout de voyage entre les sous zones, nous avons eu différentes idées :

- Calculer la distance entre les centres de gravités
- Calculer la moyenne des distances entre le centre gravité d'une zone A et toutes les cellules de la zone B.
- Calculer la moyenne des distances entre toutes les combinaisons de cellules des 2 zones.
- Calculer la plus grande distance entre deux sous zones.

La distance entre les centres de gravités à l'avantage d'être simple à calculer, cependant elle peut poser un problème dans certaines zones qui se forment par couronne successive. La distance moyenne entre le centre de gravité et toutes les cases de la zone de départ donne une estimation assez bonne et qui prend en compte l'inertie de la zone. Elle possède également la propriété intéressante de ne pas être symétrique et d'être réflexive (distance intrazone). La distance moyenne de toutes les cellules prend aussi en compte la forme des deux zones, mais reste symétrique. Enfin, la distance maximum repose sur le même calcul, mais consiste simplement à sélectionner la plus grande distance. Il serait intéressant d'essayer chacune de ces méthodes pour comparer nos résultats, mais nous n'avons pas pu par manque de temps.

Une fois ces différentes mesures décidées, nous avons réfléchi à la distance à appliquer. Plus précisément, nous avons hésité entre la distance de Manhattan et la distance Euclidienne. En effet, la carte étant formée par une grille, la distance de Manhattan nous semblait naturelle. Cependant, nous avons réalisé plus tard que le personnage pouvait se déplacer en diagonal à l'intérieur des cartes elle-même, justifiant ainsi la distance euclidienne. Finalement, nous avons appliqué la distance euclidienne pour notre optimisation.

La mer et les dimensions

Notre dernière problématique concerne la gestion de la mer et des différentes dimensions du jeu. En effet, la mer n'a pas à être traversé case par case, il nous faudrait donc imaginer une manière de prendre en compte l'impact de la mer. Ce problème est très lié à celui des transports instantanés, car nous ne pouvons pas encore récupérer la position des cartes permettant de traverser la mer. Nous avons décidé pour l'instant de calculer simplement la distance, mais c'est un problème qui devra être adressé dans la suite. Un autre problème semblable est celui des dimensions, en effet, Dofus contient différents « plan » superposé avec certain point de passage entre eux. Sans la position des points de passages, nous avons décidé d'associer une distance très élevée entre ces points, l'objectif étant donc de limiter le changement de dimension au maximum.

4.3 ASP et résultats

La représentation du problème en ASP est simple, nous allons l'expliquer en quatre parties : les prédictats, la génération, les contraintes et la minimisation.

Les prédictats

Une nouvelle fois, il nous faut réfléchir à la représentation de nos données sous formes de prédictats. Nous avons :

- `quest(id)` pour nos quêtes.
- `objective(id, quest_id, zone_id)` pour les objectifs.
- `zones(id)` pour les zones.
- `distance(zone_depart_id, zone_fin_id, cout)` pour les distances.
- `precond(quest_1, quest_2)` qui signifie que la quête 1 est une précondition de la quête 2.

Avec ces prédictats, nous sommes capables de représenter tout notre problème, il nous faut maintenant créer nos contraintes.

La génération

Il nous faut désormais générer un plan d'action. On sait que l'on aura une et une seule action par objectif de quête, on génère alors un prédictat `step(1..nb_objectifs)`. On peut maintenant écrire notre générateur : `{do(obj(I,Q), T) : objective(I,Q, _) }=1 :- step(T).`

Les contraintes

Il y a cinq contraintes à respecter :

- Contrainte de succession des étapes : s'il y a une action à l'étape n , il y a une action à l'étape $n - 1$.
- Contrainte d'unicité : il y a au maximum une action à l'étape n .
- Contrainte sur l'ordre des quêtes : si une quête 1 nécessite l'accomplissement d'une quête 2, alors on ne peut pas commencer la quête 1 tant que la quête 2 n'est pas fini.
- Contrainte sur l'ordre des objectifs : si deux objectifs se suivent dans une quête, alors ils doivent être réalisés dans cet ordre.
- Contrainte d'existence : tous les objectifs doivent être réalisés au moins une fois.

La minimisation

Nous avons déjà codé notre cout avec les distances, il nous faut maintenant calculer le cout total du modèle générée. Pour ça, on génère un prédicat `cout(valeur, etape)` qui associe à chaque étape, un cout en distance. Nous rajoutons également un cout initial entre la première sous-zone du jeu et la première quête. On utilise ensuite la directive `#sum` pour calculer un prédicat unique `total_cout(valeur)` que l'on essayera de minimiser avec la directive `#minimize`.

Résultats

Pour tester notre code, nous avons décidé d'utiliser la suite de quête de la première île du jeu *Incarnam*. Dans nos tentatives successives, nous obtenions un minimum de 6 en commençant aux alentours de 90. Par curiosité, nous avons tenté de générer tous les chemins optimaux. Cependant, le temps d'exécution est particulièrement long (plusieurs jours) et nos trois tentatives ont montré qu'il existait un nombre conséquent de chemins optimaux. Nous n'avons cependant pas eu le temps d'obtenir un nombre précis.

5 Conclusion & pistes d'amélioration

Ce projet fut très enrichissant, notamment à travers l'application de multiples méthodes de manipulation de données à un exemple à la fois insolite et enthousiasmant, d'autant plus en tant qu'anciens joueurs de Dofus. Nous avons notamment été amené à requêter une API pour récupérer nos données, utiliser une base de données SQLite pour les stocker, fait usage des nombreuses bibliothèques Python de manipulation de données et d'image comme `pandas` et `matplotlib` pour nos cartes de chaleur, ou encore `Graphviz` pour nos graphes de précédence. S'ajoute à cela la modélisation du problème en ASP avec un objectif de minimisation de distance parcourue. Ainsi, nous sommes parvenu à l'obtention de résultats tout à fait cohérents et encourageants avec les données de la zone de départ du jeu.

Nous avons également acquis une solide base de connaissances du jeu, des quêtes et des contraintes entre quêtes, et plus généralement des problématiques sous-jacentes à ce problème d'ordonnancement en particulier, ce qui s'est notamment traduit par le développement de nombreux outils réutilisables.

Bien que nous soyons encore loin d'avoir des résultats du niveau de ce qui est déjà proposé manuellement par les joueurs à travers des guides optimisés sur l'ensemble des quêtes du jeu, notre approche semble intéressante et prometteuse. La suite logique serait de l'améliorer en optimisant le calcul des distances parcourues de par la considération des transports instantanés comme nous avons commencé à le faire. Il est également possible d'envisager l'utilisation d'autres heuristiques potentiellement plus efficaces étant donné le positionnement géographique des étapes de quêtes qui n'est pas toujours renseigné ou ne rend pas nécessairement compte de la réalité des déplacements que le joueur doit effectuer.



FIGURE 5 – Heatmap de la carte principale par sous-zones

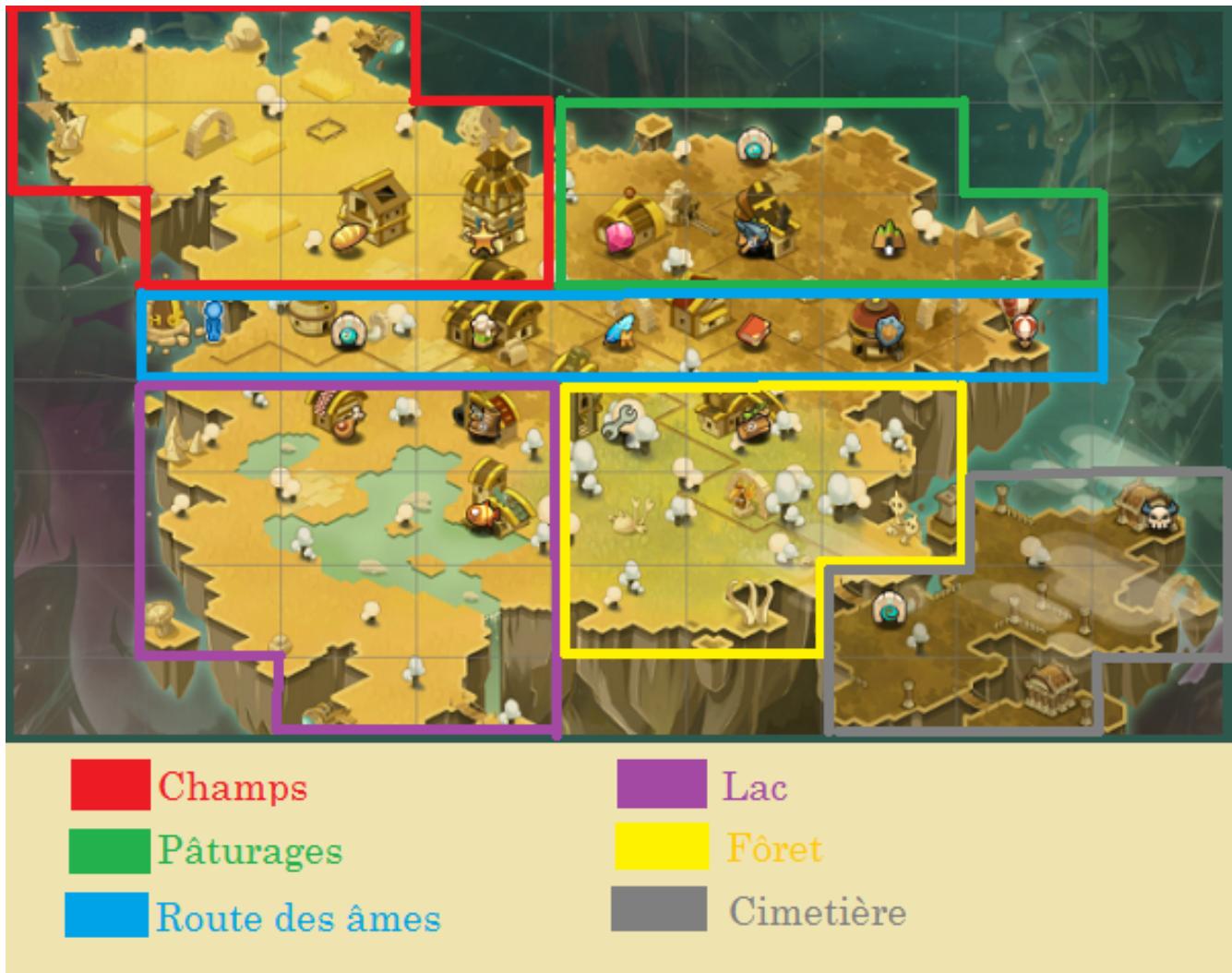


FIGURE 6 – Zone d'Incarnam

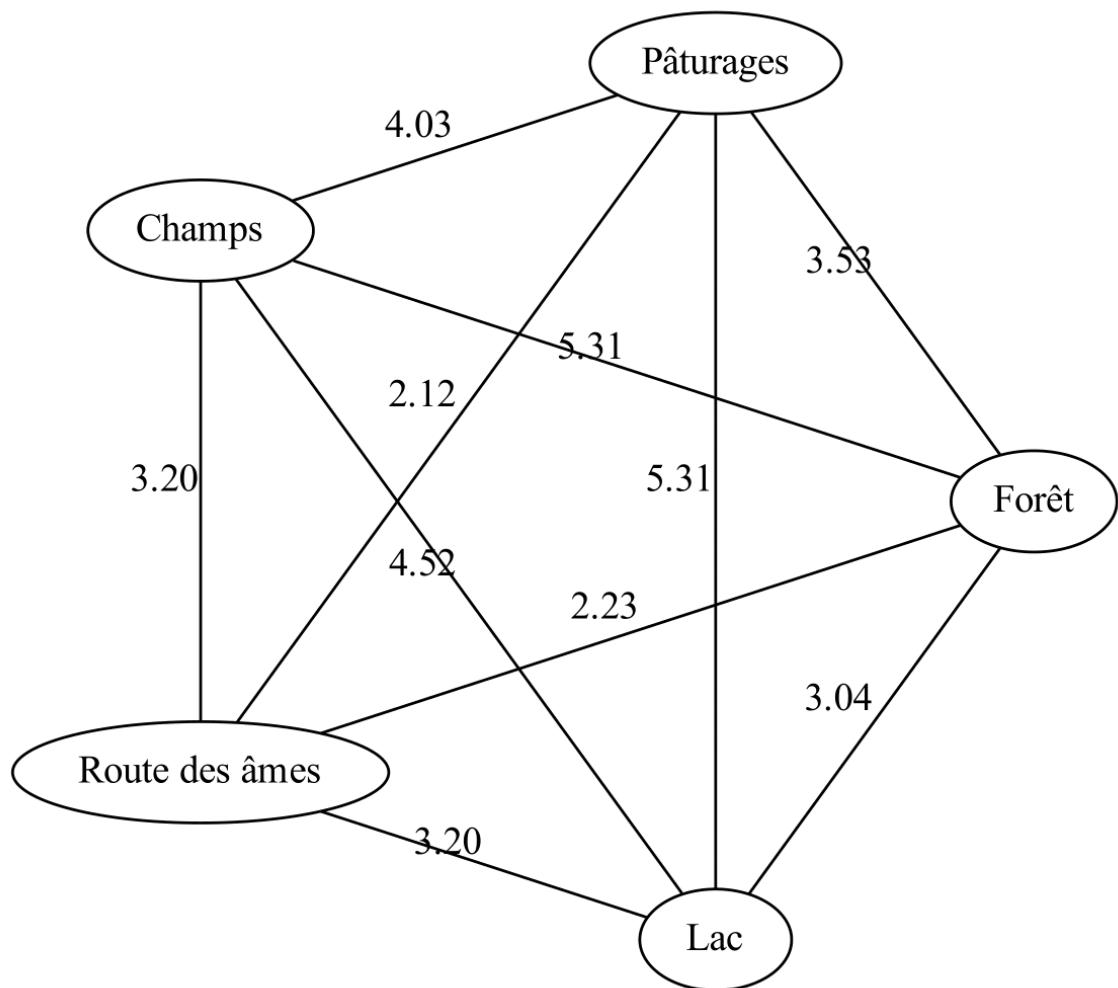


FIGURE 7 – Calcul des distances entre les centres de gravité sur Incarnam

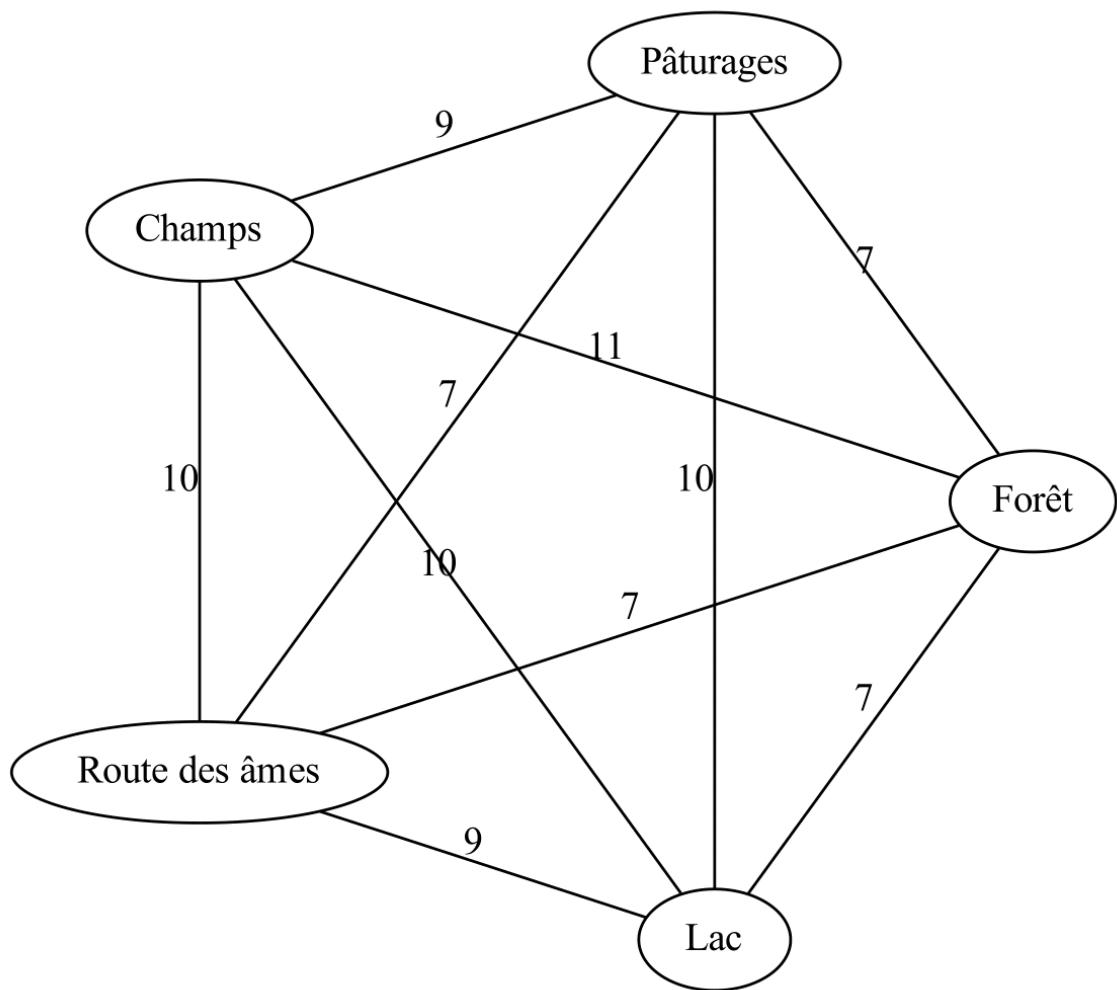


FIGURE 8 – Calcul des distances de Manhattan entre les centres de gravité sur Incarnam

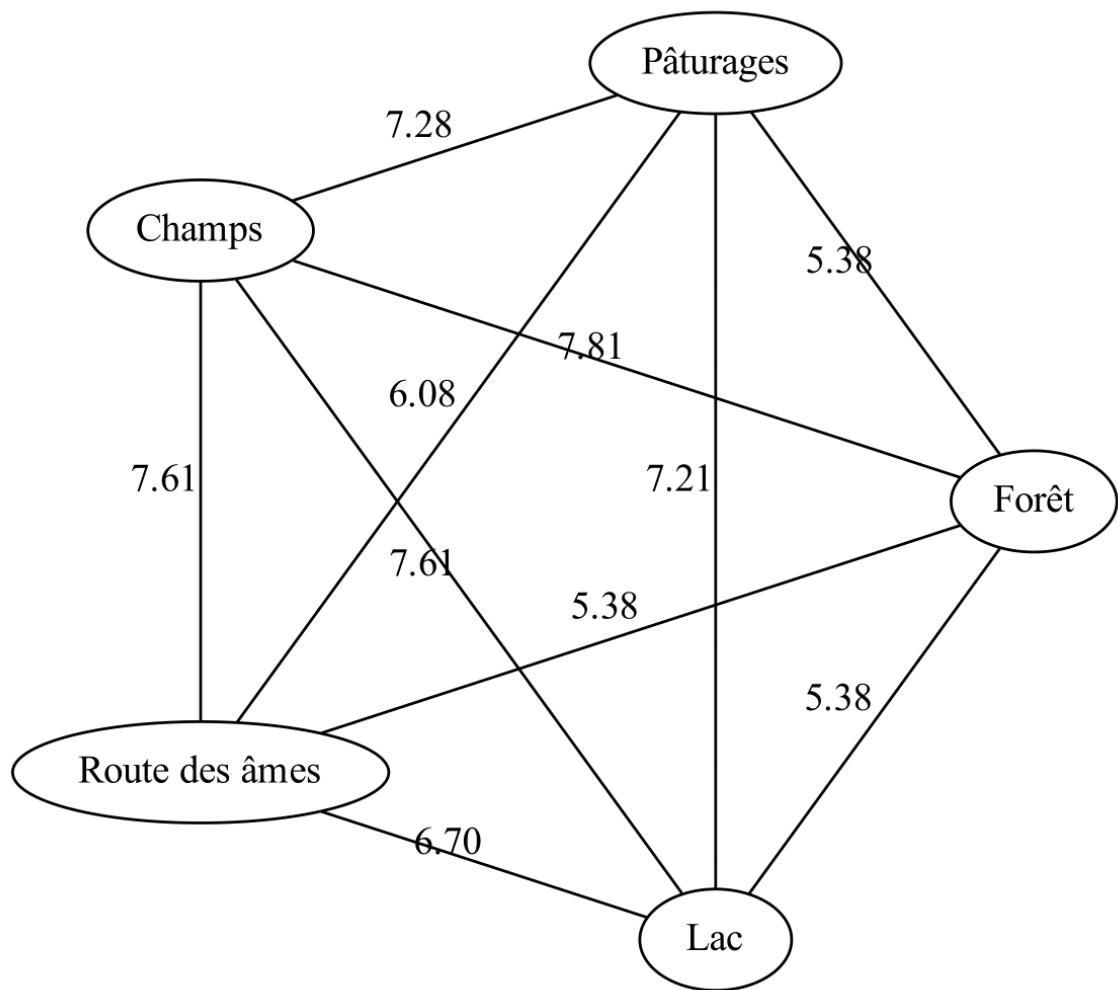


FIGURE 9 – Calcul des distances max entre toutes les cases des 2 zones sur Incarnam

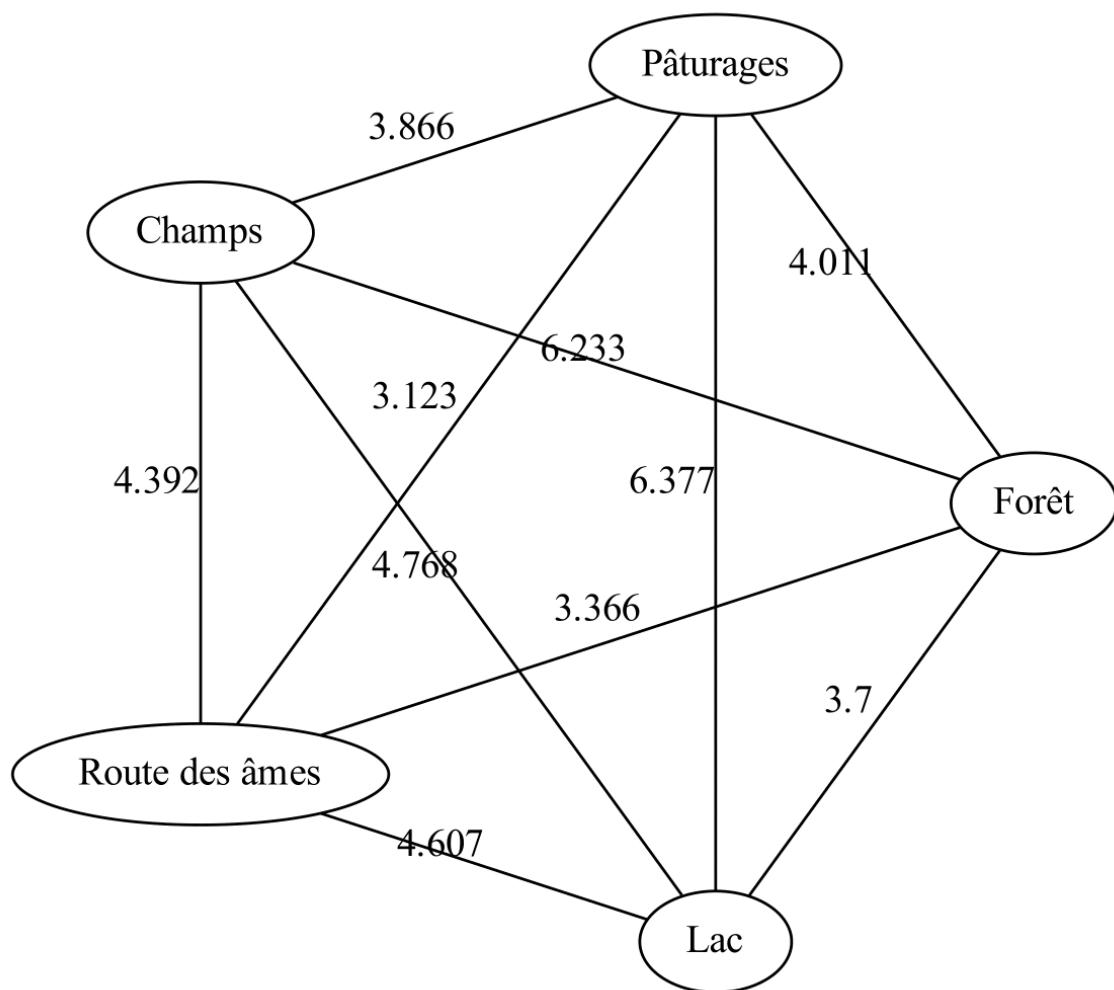


FIGURE 10 – Calcul des distances moyennes entre toutes les cases des 2 zones sur Incarnam

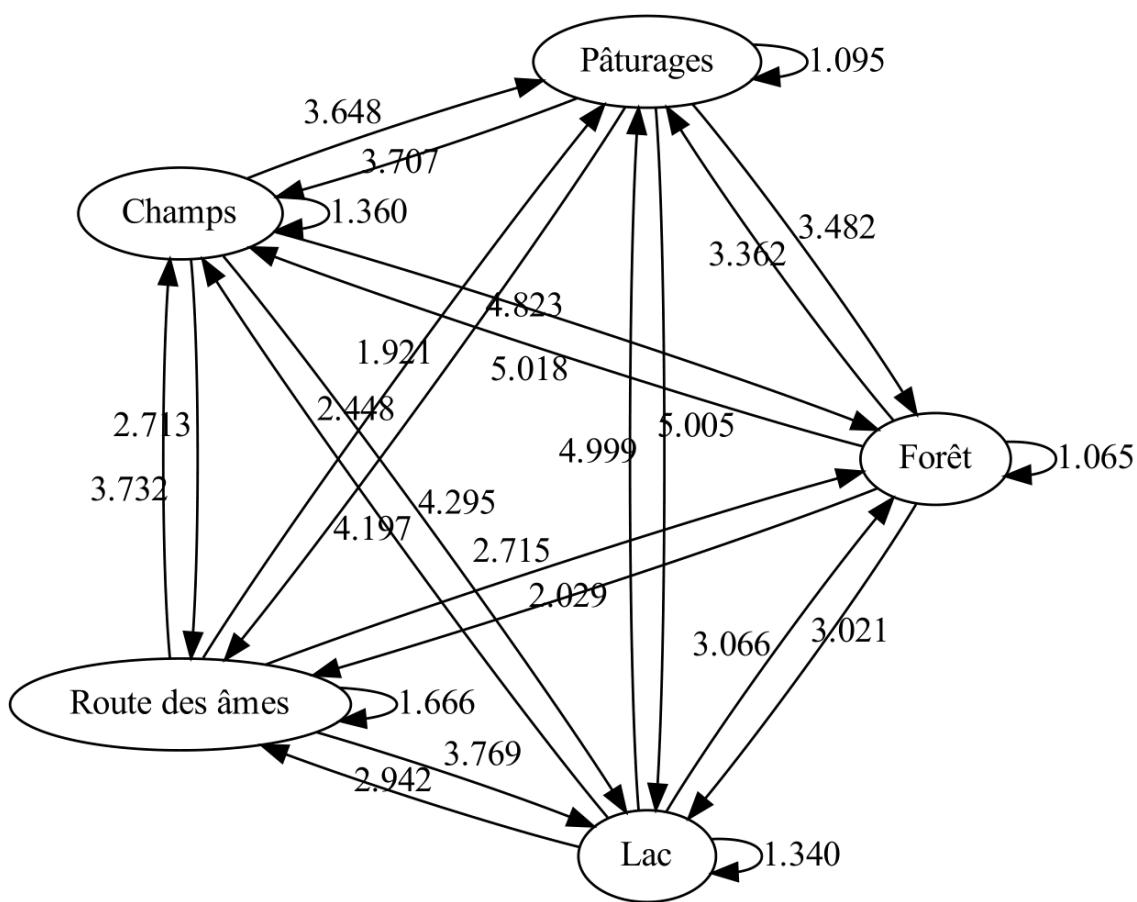


FIGURE 11 – Calcul des distances moyennes entre le centre de gravité d'une zone 1 et toutes les cases de la zone 2 sur Incarnam

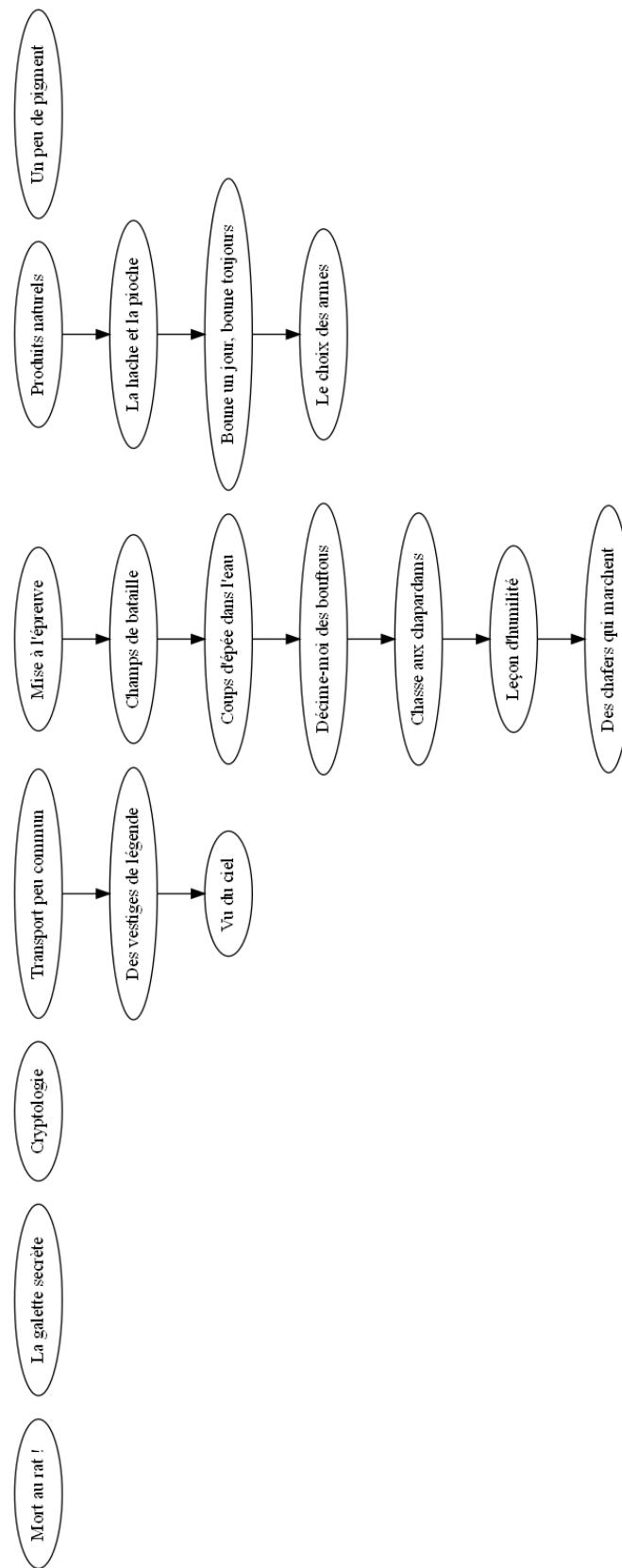


FIGURE 12 – Graphe de précédence sur Incarnam

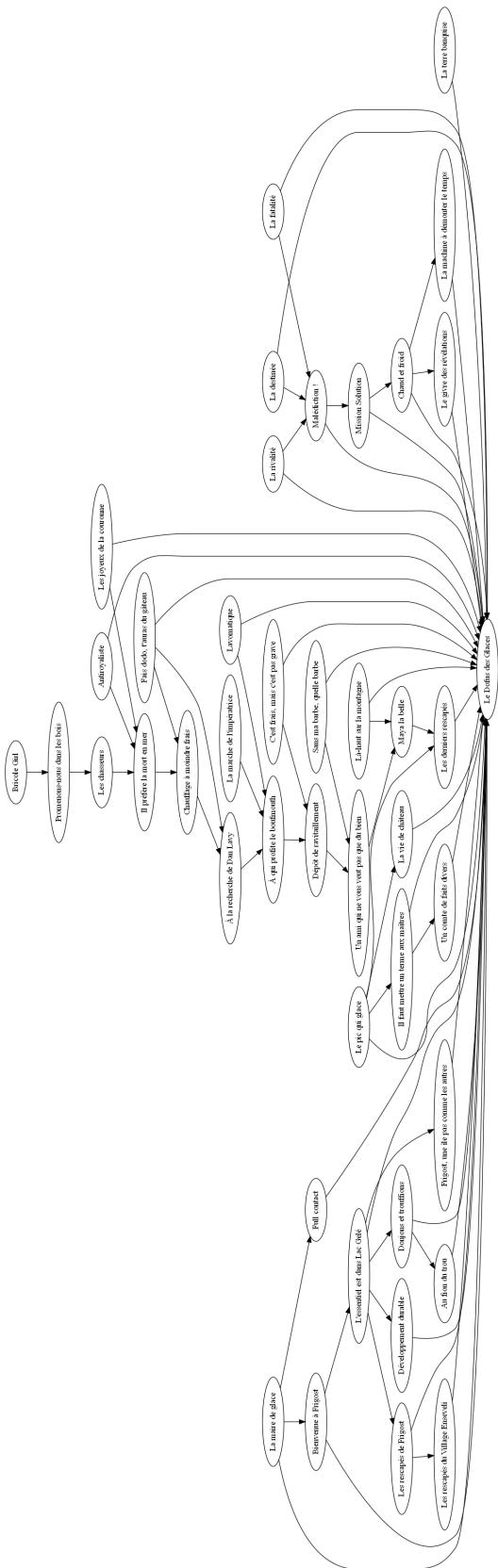


FIGURE 13 – Chemin vers la quête du dofus des glaces

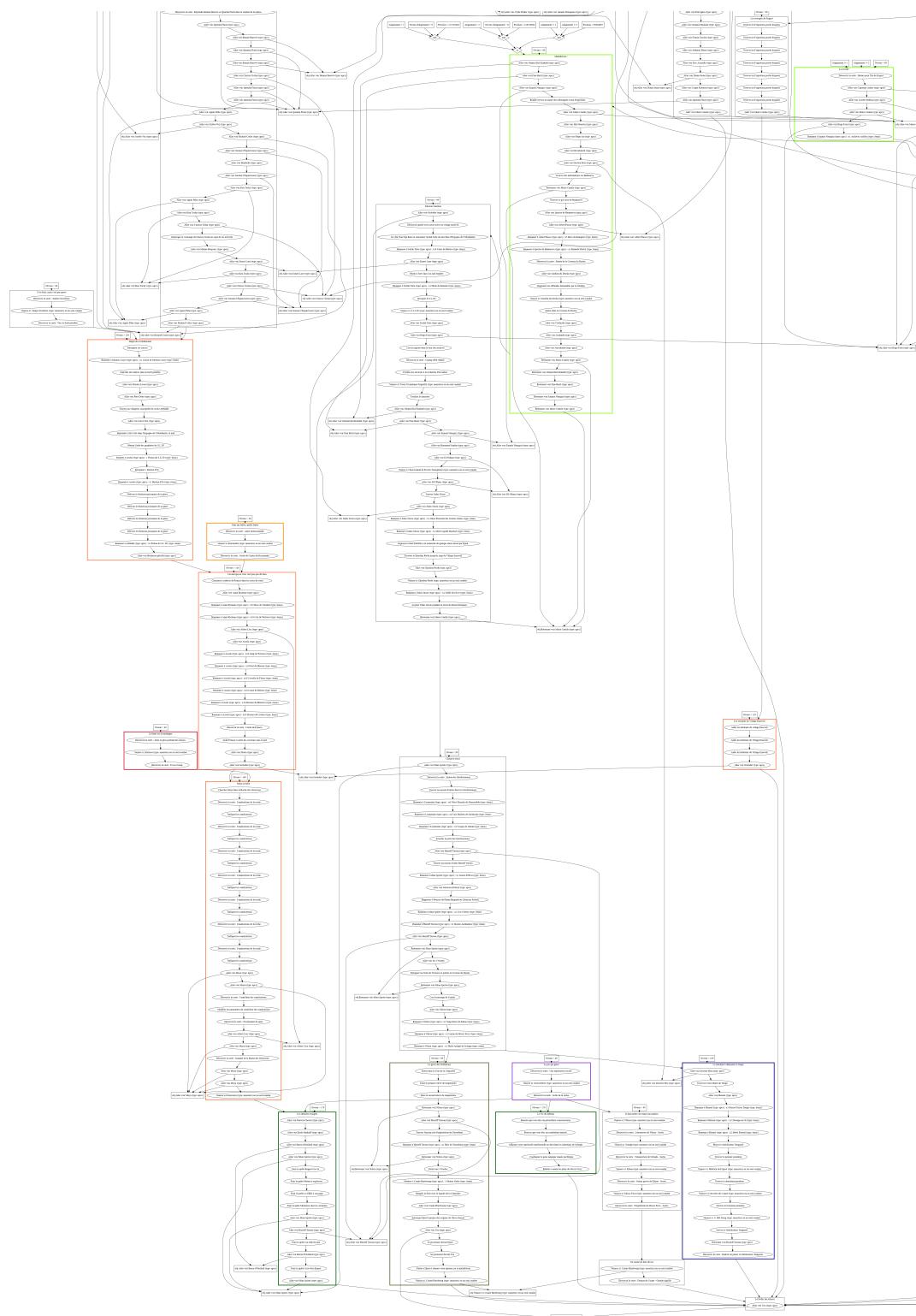


FIGURE 14 – Partie du graphe complètement développé pour le Dofus des glaces

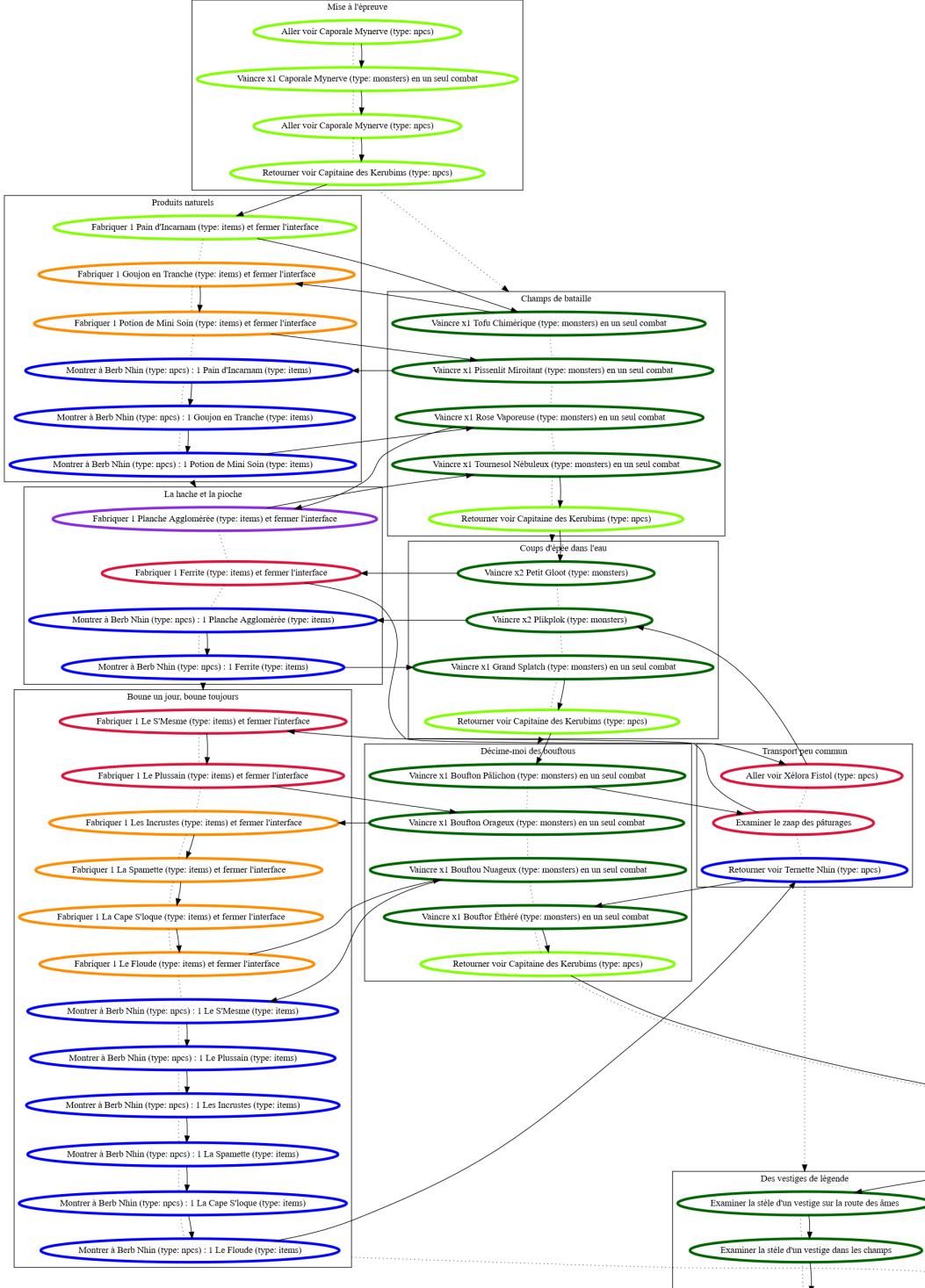


FIGURE 15 – Plan optimal pour les quêtes d'Incarnam partie 1

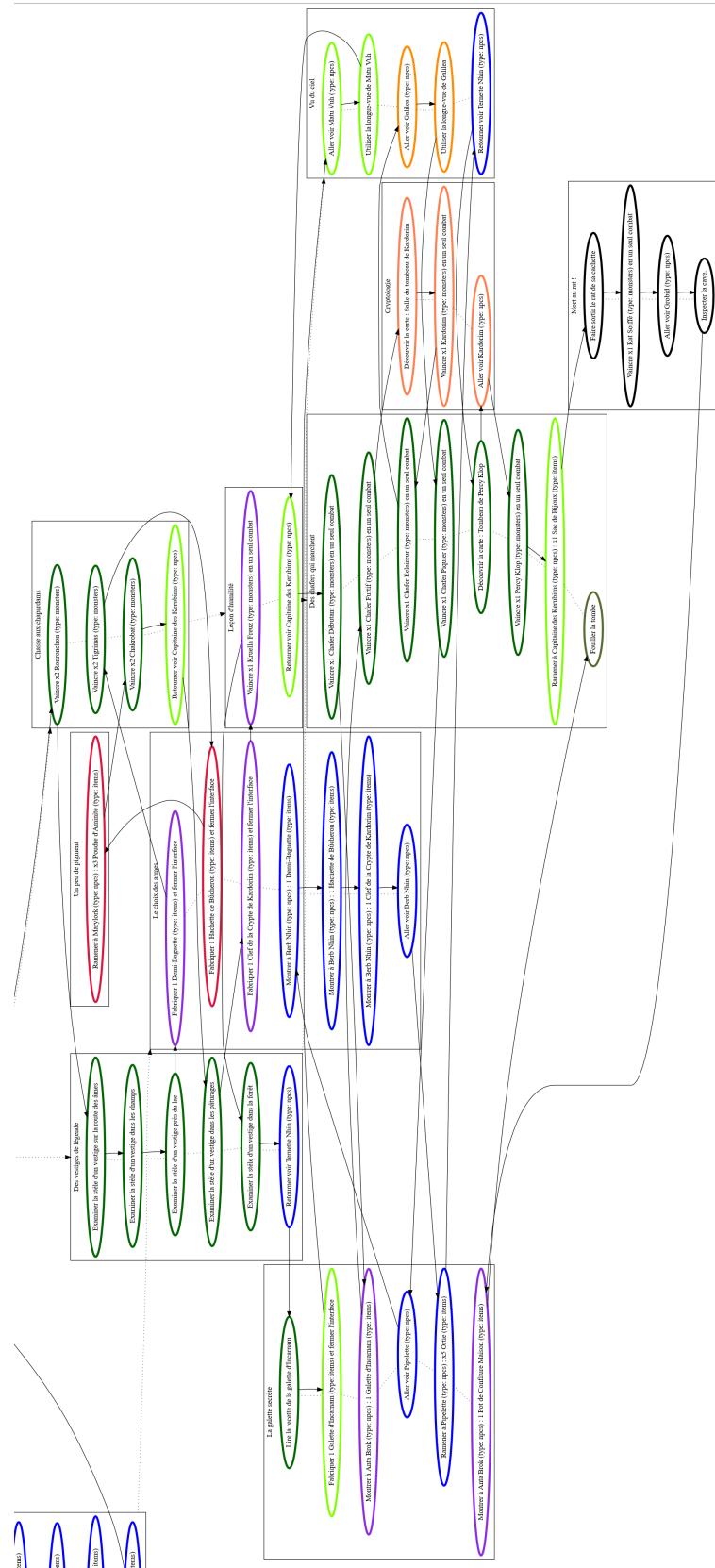


FIGURE 16 – Plan optimal pour les quêtes d'Incarnam partie 2