# Project : HGF

## Rapport de soutenance

**seconde soutenance, le 6 mars 2008**


**Félix** *Wurzag* **Abecassis** (*abecas_e*)
**Christopher** *Vjeux* **Chedeau** (*chedea_c*)
**Vladimir** *Vizigrou* **Nachbaur** (*nachba_v*)
**Alban** *Banban* **Perillat-Merceroz** (*perill_a*)

# Table des matières

# Chapitre 1

# *Introduction*

Deux mois après le premier aperçu de notre projet, La Fooo Team passe à nouveau devant le jury pour présenter l'avancée de son jeu de stratégie. Ce court laps de temps a été marqué par la mise en commun des travaux de chacun des membres de l'équipe : la mise en place d'un système de gestion de versions (SVN) a été l'élément déclencheur de ce regroupement.

Mis en place sur le serveur dédié qui nous servait déjà à héberger le site web, ce SVN nous a permis de travailler sur l'ensemble du projet, aussi bien à distance que tous regroupés dans une même pièce de 10m2. Ainsi, si chacun est focalisé sur sa tâche prédominante (Vladimir : Pathfinding, Alban : Réseau, Félix : Gestion des données, Christopher : Moteur 3D et Interface)[1], on ne peut pas déterminer exactement qui a réalisé quoi dans le projet, puisque chacun a accès et modifie l'ensemble des données du projet.

C'est un véritable travail de groupe qui a pris le dessus sur les tâches individuelles, notamment grâce à une ambiance et une entente parfaite, d'ailleurs indispensable pour surmonter les divers difficultés qui nous ont barré la route[2].

[1] Voir le rapport pour plus de détails sur les tâches citées

[2] Quelques petits problèmes se sont **pointés**

# Chapitre 2

# *Communication*

## 2.1 Travaux antérieurs

Pour assurer la promotion de notre projet, nous avions réalisé au jour de la première soutenance des fonds d'écrans, des chemises, et une vidéo d'introduction à l'effigie de la Fooo Team. De plus le site web de la Fooo Team était déjà en ligne, présentant à la fois le projet et l'équipe de production, et permettant de télécharger tous les exécutables, sources et autres produits dérivés.

## 2.2 Site web

Outre les mises à jour liées aux différentes actualités, le site de la Fooo Team (toujours disponible à l'adresse www.fooo-team.com) a subit une seule modification majeure : la mise en ligne de la version anglaise. En effet, grâce à la structure mise en place dès la création du site, les différentes localisations du site ne nécessitent plus qu'un effort de linguistique, puisqu'il suffit d'ajouter un fichier XML contenant le texte dans la langue désirée. Pour changer de langue en naviguant sur le site, il suffit de diriger le pointeur de la souris sur le drapeau de la langue en cours, et de choisir la nouvelle langue dans le petit menu ainsi apparu. Sans véritablement approfondir, la réalisation de cette petite interface m'a permis de découvrir le JavaScript, en plus de toutes les autres technologies citées dans le précédent rapport.

Sans passer par cette interface, il est possible d'accéder directement à la version anglaise du site à l'adresse suivante : www.fooo-team.com/en/index.html.

Le but du site web est de présenter le projet, mais aussi de le faire découvrir : il est ainsi nécessaire de toucher le plus de monde possible. Un bon référencement est donc nécessaire, et c'est sur ce quoi je me suis penché : d'une part faire en sorte que le site soit le mieux possible reconnu par les bots des différents moteurs de recherche grâce au respect d'une certaine sémantique, et d'autre part en relayant le lien vers de notre site sur différents autres sites. C'est une des raisons de la création du projet EpiPub.

## 2.3 Epipub

EpiPub est un projet que j'ai réalisé en partenariat avec Jean-Romain Prevost, du groupe « Freedom Fries ». Il s'agit d'une petite régie publicitaire destinée à créer un lien entre les différents sites de projets d'Epita, et plus largement tous les sites développés par les étudiants d'Epita, ayant plus ou moins de rapport avec l'école.

Il consiste en un échange de bannières entre les différents sites, affichées aléatoirement. Ainsi pour s'inscrire, il suffit de fournir trois bannières de tailles standards : 468x60, 120x300 et 180x150, au format jpg, png, gif ou même flash grâce au formulaire disponible sur le site créé pour l'occasion : www.epipub.info. Le dossier est ensuite validé manuellement pour éviter les éventuels abus. Une fois le dossier validé, le site en question n'a plus qu'à afficher au moins une des bannières sur son site grâce aux trois liens que nous lui fournissons, correspondant chacun à une des trois bannières.

Ce projet a connu un relatif succès, puisqu'à l'heure actuelle, dix projets sont inscrits, en comptant le très visité forum de la promotion 2012, affichant donc une bannière aléatoire sur la page d'accueil. Nous espérons voir s'inscrire prochainement plus de sites compte tenu que tous les groupes sont maintenant censés avoir un site web en ligne, deadline de la deuxième soutenance oblige.

Les bannières des projets inscrits actuellement sont disponibles en annexe, et consultables sur le site d'epipub : www.epipub.info.


FIG. 2.1 – Exemple de bannière leaderboard EpiPub

## 2.4 Marketing

Pour permettre à notre jury de soutenance de profiter un maximum de notre présentation, nous lui permettons de déguster un café dans les meilleurs conditions qu'il soit possible : une tasse à l'effigie de la Fooo Team. Réalisée spécialement pour l'occasion par notre imprimeur : le Studio Chaillot[1]

## 2.5 Travaux à venir

Le site web ne devrait pas subir de modifications majeures par la suite, mais à part les différentes actualités relatives à la Fooo Team et son projet.

Nous allons continuer de promouvoir le projet EpiPub, afin de rallier le plus de sites possibles à cette régie, en espérant qu'elle puisse séduire par la suite des projets de promotions différentes, et d'autres sites tournant autour de la vie Epitéenne.

Au niveau marketing, les actions se feront au fur et à mesure des différentes idées que nous pourrons développer afin de continuer de dominer le marché.

[1]Studio Chaillot, Paris 16ème, www.shirt-photo.com

# Chapitre 3

# *Interface*

## 3.1 Curseur

Le curseur lors de la première soutenance possédait deux états figés qu'étaient le pointeur normal ainsi qu'une forme de flèche lorsque l'on place la souris sur les bords de l'écran.

Un module interne de gestion des états de curseur a été développé permettant de passer d'un état à l'autre simplemenent. Dans les faits, celui-ci est utilisé pour les sorts nécessitant de cibler une unité. Le curseur se transforme alors en une cible.

Une autre amélioration apportée réside dans la mise en place de la coloration du curseur. En fonction des attributs de l'unité en dessous de la souris celui-ci change de couleur. Par exemple si on est en train de cibler un ennemi alors le curseur va être coloré en rouge.

Dans la même lancée, le curseur possède maintenant une animation. Soit directement intégrée dans la texture, soit réalisée par dégradé de couleur du noir vers le rouge par exemple.

## 3.2 Texte Affiché

Nous utilisons GLFont[1] afin d'afficher du texte à l'écran. Cet outil est constitué de deux parties. Tout d'abord un exécutable qui permet à partir d'une police de notre choix de générer un fichier contenant une image de tous les caractères ainsi que des informations relatives à la taille et la position de chacun d'eux.

Une bibliothèque quant à elle permet de lire ce fichier et d'afficher du texte. Pour un meilleur contrôle, nous l'avons recodée. Celle-ci est maintenant partie intégrante du projet. Cela permet par exemple d'afficher du texte en plusieurs polices différents.

La souplesse apportée a rendu possible des tests d'affichage en faisant varier des options afin d'obtenir un résultat lisible, mais également la possibilité d'afficher tous les caractères de la table ASCII, c'est-à-dire que les accents sont gérés.

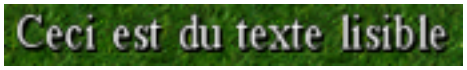


FIG. 3.1 – Exemple de texte affiché dans l'interface

Il reste toutefois encore beaucoup à faire comme la gestion des sauts de ligne automatiques lorsque l'on restreint la place horizontale ou la coloration du texte à l'intérieur d'un message.

[1] GLFont, http ://students.cs.byu.edu/ bfish/glfont.php

## 3.3 Interface

Au cours de la derniere soutenance, l'interface affichée n'était qu'un artefact. En effet, les images étaient affichées à des positions prédéterminées ce qui causait d'ailleurs des problèmes avec les différentes résolutions d'écran.

L'interface est maintenant complètement rendue grâce à l'utilisation de fichiers XML et Lua. Celà permet de détacher la forme du fond. Cette séparation oblige à se poser des questions sur la façon dont les choses vont être réalisées. Il y a également une utilité au niveau de la rapidité et la simplicité de développement.

### 3.3.1 Structure

La partie XML contient tous les éléments de l'interface. Ils sont scindés en trois groupes.

Les frames sont les éléments essentiels de l'interface. Elles ne sont pas affichées mais permettent de donner une structure. Ce sont en quelque sorte des conteneurs, ou les plans de constriuction. Chaque frame possède en son sein les objets affichés mais également une série de sous frames.

Les véritables éléments sont les Textures et les FontStrings. Ils permettent comme leurs noms l'indiquent d'afficher soit des images soit du texte. On pourrait les apparenter aux pierres de l'édifice.

### 3.3.2 Relativité

Là où cette organisation nous fait gagner du temps c'est bien dans la relativité des éléments. En effet, toute partie est relative à une autre.

La relativité commence naturellement avec la relation Parent-Enfant qui s'effectue automatiquement grâce à la structure en profondeur du XML. Elle peut également être court-circuitée à n'importe quel endroit de la chaîne.

C'est dans son utilisation que réside son sens. Chaque élément est placé en fonction d'un autre. Dans l'image ci-dessous, la première icône a été placée par rapport au bord inférieur droit de l'écran. Chaque icône qui suit est ensuite positionnée à quelques pixels à droite ou en bas de son voisin.

Une fois le premier élément placé il devient un jeu d'enfant de placer les autres. Les modifications spatiales deviennent grâce à ce système extrêmement rapide et intuitives.

FIG. 3.2 – Position relative des éléments par rapport au premier

### 3.3.3 Héritage

Il est possible de créer des frames qui ne seront pas affichées mais qui servent de modèle. Des frames, bien réelles cette fois-ci, vont être capables d'hériter de toutes les propriétés et objets du modèle. Ainsi afficher par exemple plusieurs boutons, ayant un fonctionnements complexe, est l'affaire de quelques lignes.

L'héritage est une base pour l'élément, toutes les propriétés héritées peuvent être remplacées selon l'utilisation spécifique. Cela donne l'avantage d'établir des composants généraux qui peuvent être réutilisés facilement.

## 3.4 Interactivité

Afin de rendre interactif cette interface nous avons fait le choix d'utiliser le langage de script Lua. Il est extrêmement simple à mettre en place et dispose d'une importante flexibilité qui permet de coder rapidement. N'étant pas compilé avec le programme, il est possible de prendre en compte les modifications sans avoir besoin de relancer le jeu.

Un système d'évènement a été mis en place afin de pouvoir appeler les fonctions programmées en Lua. On associe à chaque frame des évènements comme OnLoad, OnKeyDown ou encore OnMouseEnter. Il existe également OnEvent qui est appelé lors d'évènements relatifs au déroulement du jeu tel que l'arrivée d'un joueur.

## 3.5 Application

Afin de mettre en pratique le framework nous avons dans un premier temps réalisé une barre de texte ainsi que l'affichage de messages. Maintenant que les bases sont posées nous allons pouvoir nous consacrer au développement des modules de l'interface.


FIG. 3.3 – Barre de texte où l'on écrit du texte

# Chapitre 4

# *Moteur 3D*

## 4.1 Optimisations

Les possibilités du moteur 3D n'ont pas été etoffées, mais un effort d'optimisation a été fait. C'est après le constat désolant d'un jeu qui n'était plus fluide avec seulement deux unités affichées que l'on s'est intéressé plus en profondeur aux améliorations possibles.

### 4.1.1 Frustum Culling

La première optimisation évidente est de ne pas afficher les unités si elles ne sont pas présentes dans le champs de vision. Alors que nous avions déja un bout de code qui réalisait cette tâche de façon empirique, il a fallu faire appel aux mathématiques afin de trouver une solution.

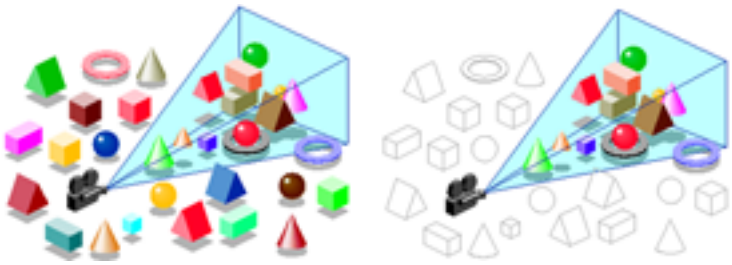

FIG. 4.1 – Elimination des modèles non visibles

Le problème est simple, on veut tester si un modèle est présent dans le champs de vision. Tout d'abord, il faut savoir que le champs de vision est constitué de 6 plans. On se retrouve donc à chercher si un modèle est inclu dans l'espace formé par ces 6 plans.

La solution parfaite serait de déterminer si chaque triangle est visible ou non. Ainsi on afficherait uniquement les triangles visibles.

Le problème étant que le test de présence a un coût machine non négligeable par rapport à l'affichage d'un triangle. Donc on a pour idée de tester si une boîte contenant le modèle est présente dans le champs de vision. C'est à dire tester 8 points. Le problème c'est que les fichiers modèles ne contiennent pas les informations d'une potentielle boîte qui contiendrait le modèle.

Nous avons uniquement à disposition les coordonnées du diamètre d'une sphère qui contient le modèle entier. Après des recherches il apparaît que le test de visibilité d'une sphère est très peu couteux en opérations comparés à celui d'un cube. Nous voici donc avec une élimination des modèles non visibles.

### 4.1.2 Vertex Array

La seconde grosse optimisation réside dans la façon d'afficher les polygones. Au départ nous utilisions la méthode traditionnelle d'OpenGL qui consiste à envoyer naïvement les coordonnées des vertex un par un à la carte graphique. Après de nombreuses recherches il existe plusieurs options pour accélérer le rendu. Celles-ci ont toutes pour principe d'envoyer les coordonnées des vertex groupées dans un tableau. C'est dans la disposition des données que divergent les méthodes.

Dans notre cas, nous allons d'abord envoyer l'adresse d'un tableau énorme contenant toutes les donnéees affichables, puis donner un tableau plus petit contenant uniquement les indices des éléments à afficher. De cette façon la carte graphique va pouvoir récupérer les informations beaucoup plus rapidement.

Nous nous sommes également aperçu que pour le terrain et l'interface nous n'utilisions que deux coordonnées au lieu des trois. En effet, le terrain ainsi que l'interface n'ont pas de hauteur. Ainsi en retirant ce paramètre des éléments envoyés à la carte graphique nous avons pu obtenir des bénéfices notables en terme de performance.

## 4.2 Picking

### 4.2.1 Terrain

Le picking était déja fonctionnel à la première soutenance mais ne concernait que les carrés de textures du terrain. Pour récupérer la position de la souris par rapport au terrain nous utilisons une autre méthode. Au lieu d'utiliser le picking nous utilisons encore des mathématiques. OpenGL met à disposition une fonction qui permet de récupérer les coordoonnées de deux points de la droite projetée à partir de la souris vers le terrain. Il suffit ensuite de calculer l'intersection de cette droite et du plan d'équation z=0.

### 4.2.2 Models

Pour les modèles la techniques du picking par OpenGL fonctionne très bien. Dans un souci d'optimisation nous effectuons également un test de champ de vision mais cette fois-ci sur un carré de dimensions 1x1, ou plus pour un rectangle de selection.

La prochaine étape concernant le Picking est de le faire fonctionner au niveau de l'interface.

# Chapitre 5

# *Gestion des données*

## 5.1 Ce qui a été realisé

Lors de la première soutenance, j'ai présenté trois exécutables, un outil de modification des options du jeu, un éditeur d'unités et de compétences et une simulation de gestion des données en temps réel, le tout en TForm et basé sur des données stockées sous format XML.

Ces deux premiers outils, totalement indépendants du jeu stricto sensu, n'ont pas été modifiés pour l'instant, mais cela n'exclut pas de nouvelles modifications dans le stade final du développement en fonction des nouvelles options qui seront disponibles pour le jeu et des nouveaux attributs personnalisables pour les unités et les compétences. Ces ajouts devraient se faire sans aucune difficulté puisque la structure principale est déjà pleinement fonctionnelle, ces modifications se résumeront à quelques ajouts de paramètres dans le code.

Quant au dernier outil présenté, il n'était bien sur que temporaire, il s'agissait pour moi de chercher une structure de fonctionnement pour une gestion de données en temps réel comprenant, entre autres : la gestion des équipes, le déplacement, les points de vie, les créations et destructions d'unités, la gestion des effets des compétences et de leur durée, etc.

Cet essai m'a permis de réfléchir et de décider d'un paradigme de gestion en pouvant aisément voir les conséquences de mes modifications grâce a la simplicité de manipulation du TForm puisque la gestion de données s'affranchit par essence du moteur graphique.

## 5.2 Les nouvelles fonctionnalités

### 5.2.1 Introduction

Néanmoins, conscient du fait que mon modèle de gestion devrait être confronté à la réalité du jeu, la convergence vers le travail de Christopher, c'est-à-dire l'union de la gestion de données et du moteur graphique, s'est promptement amorcée dés la reprise du travail après la première soutenance.

La finalité permanente recherchée lors de notre travail coopératif a été de développer une interaction intuitive et performante entre le jeu et l'utilisateur.

Mon but premier était d'intégrer dans le jeu toutes les possibilités offertes par la gestion de données en TForm tout en surmontant les éventuelles difficultés rencontrées lors de l'adaptation de mon modèle de gestion abstrait à un nouveau mode de fonctionnement plus visuel où la base de l'interaction est la sélection de une ou plusieurs unités.

### 5.2.2 Affichage

La première étape accomplie fut de pouvoir créer une unité en désignant sa position sur la carte de jeu avec le curseur de la souris tout en conservant en mémoire ses caractéristiques chargées depuis un fichier XML.

Puis, les unités sont devenues sélectionnables grâce au picking réalisé par Christopher, l'unité sélectionnée arborant un cercle de sélection de couleur à ses pieds.

FIG. 5.1 – Cercle de sélection

### 5.2.3 Interactions

Cette nouvelle possibilité a ouvert la voie à tous les autres types d'interactions possibles sur des unités en affichant sur l'interface les caractéristiques de l'unité sélectionnée, ce qui s'avère indispensable pour mener à bien des tests afin de vérifier le bon fonctionnement de la gestion de données.

Une interaction fondamentale dans tous les jeux de stratégie est le déplacement des unités, sans qu'il soit question ici de recherche de plus court chemin, mon objectif était de pouvoir déplacer une unité sélectionnée par un simple clic droit sur une position inoccupée. Ce déplacement se faisait en ligne droite par simple translation du modèle à une célérité dépendante de la vitesse de déplacement de la dite unité, bien sur, en cas d'obstacle sur son chemin, l'unité était incapable de le contourner. Afin de rendre le mouvement plus crédible, la rotation des unités vers la position cible a ensuite été progressive.

L'étape suivante a été de permettre les interactions entre unités, en commençant par l'attaque : une unité sélectionnée peut attaquer une unité cible en la désignant par un clic droit, l'unité attaquante se met alors à se déplacer vers sa proie et même à la suivre si elle tente de fuir pour finalement lui infliger des dégâts une fois à portée d'attaque, et si elle subit plus de dégâts qu'elle n'a de points de vie actuellement, elle meurt et disparaît du jeu.

### 5.2.4 Compétences

La gestion de l'utilisation des compétences dont les effets sont stockés dans un fichier XML a été réalisée ensuite. Elle gère son coût en mana, la distance de lancement, et empêche que le même effet s'additionne plusieurs fois sur une unité. Les compétences disponibles pour la démonstration sont : Furie Sanguinaire (augmente l'attaque et baisse la défense) avec la touche F4, et rapidité (le nom est suffisamment explicite) avec la touche A. Il est possible de remplacer ces compétences par n'importe lesquelles au vu de la souplesse de la gestion des effets des compétences.

Je me suis ensuite attelé à la création de bâtiments : en appuyant sur la touche S et en choisissant un emplacement, l'unité sélectionnée se déplace à la position voulue et commence à construire le bâtiment. Tant que cette unité est à proximité du bâtiment, il se construit petit à petit en fonction du temps de construction prédéterminé dans le fichier XML, mais si elle s'éloigne, la construction est momentanément interrompue. Coté interface, vu qu'il est bien entendu impossible de créer un bâtiment sur une zone déjà occupée, un carre de validité s'affiche lors du choix de l'emplacement et se colore en vert ou en rouge selon la validité du lieu.

Dans l'exemple ci-dessous, l'unité sélectionnée ne peut pas construire à l'emplacement désiré car le bord inférieur gauche est déjà occupé par un autre bâtiment.

FIG. 5.2 – Impossibilité de construire sur un bâtiment

### 5.2.5 Sélection

L'avancée qui a suivie fut capitale, en se basant sur les fonctionnalités ajoutées jusqu'ici, j'ai introduit la gestion des équipes : une unité ne peut pas attaquer un allié mais elle peut le suivre, certains sorts ne peuvent être lancés que sur certaines cibles particulières (alliés, ennemis ou soi-même), etc.

Cette distinction a ensuite permit l'amélioration de la sélection à la souris et au clavier, la multi-sélection est devenue possible grâce a un rectangle de sélection crée par un clic gauche et un déplacement de la souris, les unités de notre équipe situées sous le rectangle sont ajoutées à la liste de sélection. Toutes ces unités peuvent alors agir de concert, par exemple, elles se déplaceront ensembles vers une position ou une unité cible.

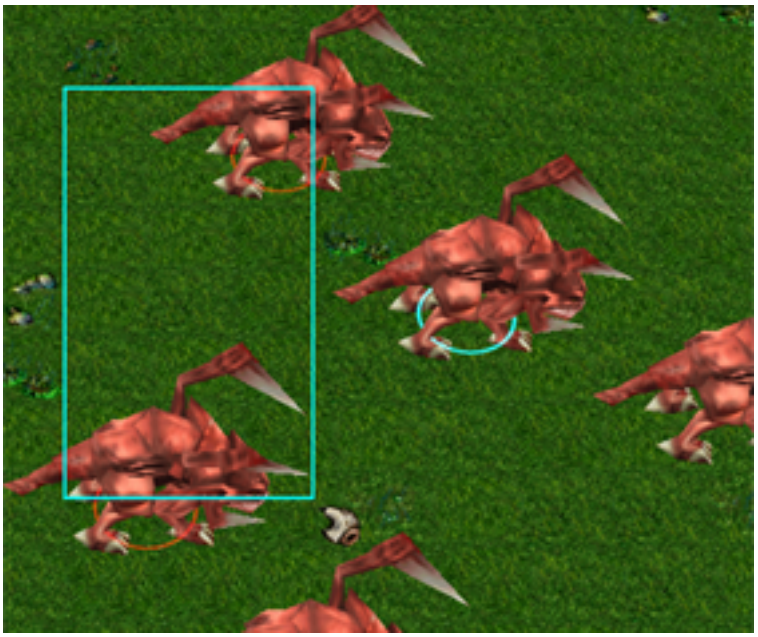

FIG. 5.3 – Rectangle de sélection

Pour faciliter la manipulation des unités du joueur, il est possible de se souvenir du groupe d'unités sélectionnées en utilisant la combinaison de touches CTRL+0..9 pour ensuite pouvoir rappeler ce groupe en utilisant une des touches 0..9. Une unité peut être rajoutée à un groupe de sélection en utilisant la combinaison SHIFT+0..9.

Enfin, toujours d'un coté pratique, l'utilisation du double clic ou de CTRL+clic sélectionne toutes les unités de son équipe et du même type à proximité.

### 5.2.6 Intégration

Un grand travail de concertation s'est ensuite déroulé en symbiose avec Vladimir dans l'optique d'intégrer son algorithme de recherche du chemin le plus court dans le jeu.

Il a fallu pour cela remodeler certains aspects de la gestion pour pouvoir permettre le déplacement des unités par points de passages comme exigé par l'algorithme A star.

Le plus gros changement apporté fut un tout nouveau système générique et flexible d'ordres permettant d'ajouter facilement de nouvelles fonctionnalités et de gérer de manière analogue les déplacements, les attaques, les sorts,

les constructions, etc.

Cela a clarifié grandement le code, facilitant ainsi notre collaboration. L'ajout de son algorithme A star dans le projet est ensuite devenu possible.

Concernant l'échange de données par le réseau, comme toutes les données sont stockées dans des fichiers XML et donc facilement modifiables, il est apparut nécessaire de vérifier que certains fichiers sont identiques pour tous les joueurs. Cependant il n' est pas exclu que certaines données soient imposées par l'hôte de la partie.

Pour cela, nous avons intégré une fonction de hachage MD5 dans le jeu, elle aura pour rôle, en admettant que les risques de collisions sont négligeables, de comparer les fichiers au lancement de la partie pour éventuellement exclure les joueurs présentant une version modifiée.

## 5.3 Bilan intermédiaire et objectifs

Je suis satisfait du travail accomplit entre ces deux soutenances, j'estime avoir travaillé avec sérieux et ce travail a été récompensé puisque la gestion des données dans le jeu est à mon avis bien avancée, mieux encore que je ne l'aurais espéré au lendemain de la première soutenance. La collaboration avec les autres membres du groupe s'est très bien passée, même si la mise en commun a été difficile puisque cela impliquait de bien assimiler ce que les autres ont réalisé afin d'adapter sa propre partie et éviter les collisions.

Pour la suite, il est nécessaire de continuer la convergence avec la partie réseau pour assurer la synchronisation en temps réel de toutes les données du jeu afin de permettre le jeu multi-joueurs.

Je reste confiant pour la suite, nous avons tous accomplit du bon travail jusqu'à maintenant, et cela devrait et doit continuer. De plus, la fusion avec la partie de Christopher étant maintenant effectuée, je n'aurais plus de travail d'adaptation à réaliser, ce qui me laissera le champ libre pour me focaliser sur le développement de ma partie.

# Chapitre 6

# *Réseau*

## 6.1 Travaux antérieurs

Pour développer le réseau du projet HGF, j'avais choisit la bibliothèque Indy 10, intégrée à Delphi. J'avais réalisé grâce à l'outil TForms une simulation de réseau sous forme de « chat », en envoyant simplement des chaînes de caractères via le réseau.

## 6.2 Abandon d'Indy

L'utilisation d'Indy s'est rapidement compliquée pour moi : la documentation s'est avérée inefficace et la multitude de composants disponibles ne faisait que de noyer les outils que j'utilisais vraiment.

J'ai donc choisis d'utiliser directement le composant Winsock, intégré à Windows. Celui-ci comporte plusieurs avantages à mes yeux : sa légèreté le rend simple d'utilisation et me permet de comprendre le fonctionnement de chaque fonction que j'utilise. Sa portabilité me permettra par la suite de réutiliser mes connaissances acquises dans différents langages et sur différentes plateformes. Enfin ce composant est très bien documenté, autant dans sa version Windows que Unix.

## 6.3 Intégration du réseau dans le jeu

Après avoir rapidement posé les bases du réseau dans une application console, je me suis attaqué notamment grâce au SVN à l'intégration du réseau dans le jeu. J'ai pu dans un premier temps tester l'envoi de messages grâce à l'affichage de texte sur la gauche de la fenêtre de jeu. Par la suite, l'arrivée de

la saisie de texte m'a permis de mettre en place un véritable chat permettant d'envoyer du texte personnalisé.

Cependant le chat étant opérationnel, j'ai du m'intéresser à la structure du réseau : les données étaient pour l'instant directement envoyées selon une adresse IP, et il fallait mettre en place un système de serveur auquel les clients se connectent, pour centraliser les informations. Contrairement à ce que je pensais, il fut plus simple de faire un serveur intégré au client et non dédié. Les joueurs choisissent un serveur auquel se connecter, et celui-ci est chargé de recevoir toutes les informations de chaque joueur et de les diffuser aux clients intéressés.

Pour envoyer des données autres que du texte, il m'était intéressant de pouvoir envoyer des types structurés par le réseau. Le composant Winsock le permet très facilement : je peux ainsi identifier le type de données reçues par le serveur à l'aide du premier champ du type structuré et donc le traiter en conséquence.

Il m'est donc pour le moment possible, en plus du chat, d'envoyer l'ordre de création d'une unité par le réseau. Les unités affichées à l'écran d'un joueur sont distinguées par un code de couleur : les unités bleues sont celles du joueur, les rouges sont celles des joueurs adverses. Il est aussi possible d'infliger des dégâts à une unité via le réseau, et ainsi la faire disparaitre si celle-ci venait à mourir.

## 6.4 Travaux à venir

Le plus gros du réseau a maintenant été implémenté. Il me faudra maintenant faire en sorte que toutes les actions puissent être transmises via le réseau : déplacements, sorts, etc.

# Chapitre 7

# *Déplacements d'unités*

## 7.1 Recherche de chemin

Comme convenu au début du projet, la base du système de recherche de chemin repose sur l'algorithme de pathfinding A*. Comme nous allons le voir, cet algorithme ne représente pas la partie intégrale de la recherche d'un chemin mais n'est qu'une partie de la solution.

### 7.1.1 Améliorations du chemin

A cette heure, nous utilisons toujours une représentation du monde par une matrice de cases de même taille, assimilable à un damier sur lequel chaque case est accessible soit par ses voisins directs, soit en diagonale, ce dernier déplacement étant plus long.

Quelques améliorations vraiments basiques de la recherche ont vite été ajoutées. La première a été au lieu de chercher non pas à aller au point d'arrivée, mais seulement au point le plus proche, ce dernier pouvant évidemment être le point d'arrivée désiré comme au pire des cas le point de départ. Cette amélioration permet par exemple d'autoriser le joueur à cliquer d'une manière imprécise lorsqu'il souhaite effectuer un mouvement général dans une direction plutôt que d'atteindre un point précis.

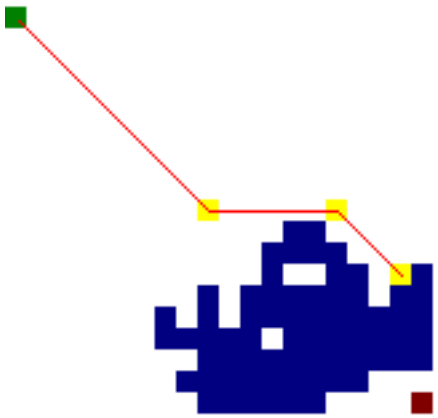

FIG. 7.1 – L'arrivée (en rouge) étant inaccessible, l'algorithme se rapproche le plus possible

Une autre optimisation de l'algorithme consiste à ne donner que la liste des points pivots du chemin, c'est à dire les points de changement de direction. En effet, pourquoi ordonner par exemple de se déplacer 4 fois d'une case à droite quand il suffit de se déplacer à droite, de 4 cases ? Dans notre cas, la liste des points pour aller d'un coin à l'autre d'une carte carrée vide se réduit simplement au départ et à l'arrivée, puisqu'il est possible d'y aller directement en ne suivant qu'une seule des 8 directions de base. La réduction du nombre de points de passage donnés par la recherche de chemin permet ainsi dans le jeu de diminuer la mémoire qu'une unité prendra pour stoquer son chemin.

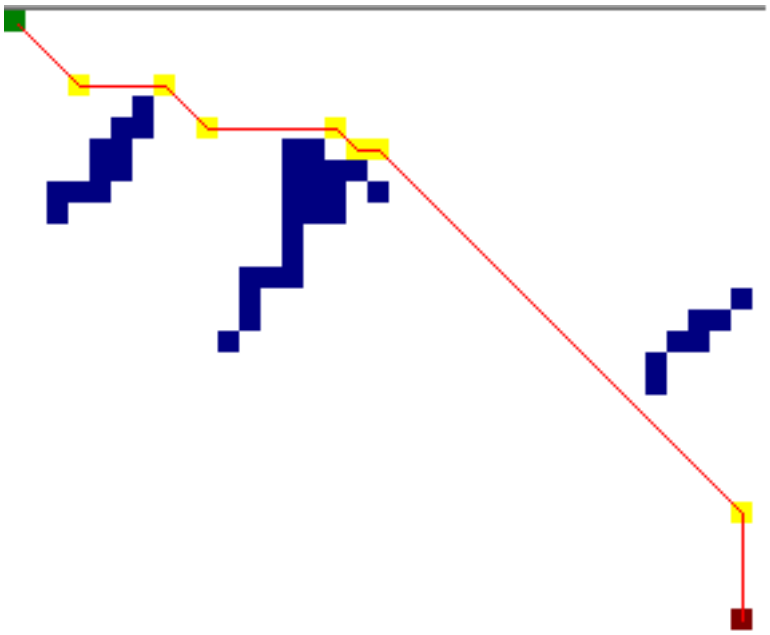

FIG. 7.2 – Au lieu de lister toutes les cases du chemin, l'algorithme ne donne que des points pivots (en jaune)

Enfin, dans un jeu tel que le nôtre, les unités et même les obstacles n'ont pas tous la même taille qui pourrait servir par exemple de taille standard de case. Il faut donc prévoir que l'algorithme ne cherche pas forcément un chemin prévu pour une case, mais qu'il évite par exemple de passer entre 2 obstacles trop rapprochés.

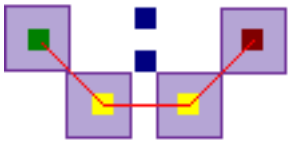

FIG. 7.3 – Ici, l'unité ne peut pas passer entre les 2 obstacles, l'algorithme les contourne donc

### 7.1.2 Amélioration du temps d'exécution

La recherche de chemin est un point fondamental du jeu, il sera par exemple appelé lorsque le joueur cherchera à déplacer son unité, mais aussi automatiquement par exemple lorsqu'une unité en suivra une autre. Il est donc nécessaire pour ne pas ralentir le jeu d'optimiser les algorithmes qui déterminent un chemin. Le principe théorique de l'A* est plutôt simple, reposant sur une "liste ouverte" et une "liste fermée", il faut très souvent trouver le meilleur noeud de la liste ouverte, et en plus de celà il faut constamment voir si un noeud adjacent à celui en cours de traitement est dans une de ces deux listes. Il faut donc trouver une implémentation adaptée pour représenter ces listes afin d'avoir une complexité minimale.

Le problème de l'accès aux noeuds adjacents et de la vérification de leur appartenance à une liste peut être reglé facilement en maintenant une matrice de la taille de la carte contenant un pointeur vers un noeud, en plus d'une simple variable précisant la liste dans laquelle elle est. Ainsi on peut déterminer si un noeud voisin est dans une liste en temps constant. Cette solution pourrait poser le problème suivant : à chaque noucelle recherche de chemin, il faudrait parcourir chaque case et préciser qu'elle n'est dans aucune liste. Ceci est evité simplement en utilisant un entier pour déterminer la liste : en effet, en maintenant une variable entière et en l'augmentant à chaque fois de 2, on peut l'utiliser pour représenter la liste ouverte, fermée, et considérer qu'une autre valeur n'est pas encore dans une liste. Finalement, la matrice contenant un noeud et un statut n'a qu'à être initialisée au chargement d'une carte.

Il faut aussi pouvoir déterminer le meilleur noeud de la liste ouverte, soit celui aillant le coût le plus faible. Pour celà, l'utilisation d'une autre liste est nécessaire, en effet, il serait inimaginable de parcourir une matrice de la taille de la carte entièrement afin de déterminer le meilleur noeud. Il faut donc utiliser une file de priorité, la priorité la plus haute étant donnée au noeud ayant le coût le plus faible. Mon choix à ce point s'est porté sur un arbre binaire de recherche équilibré. Cette structure permet l'insertion d'un élément tout comme l'extraction de l'élément à la plus haute priorité avec une complexité logarithmique.

Finalement, bien que la complexité nous permette de nous faire une idée, le plus important est la rapidité réelle de l'algorithme pour trouver un chemin. Toujours en utilisant mon interface en Tform Delphi, j'ai donc rajouté une option permettant d'effectuer des tests de performance, en mesurant le temps d'exécution moyen d'une recherche, en effectuant cette dernière un certain nombre de fois. Il sera donc plus simple s'il s'avère que l'algorithme est trop lent pour le jeu, de travailler l'optimisation de ses performances.

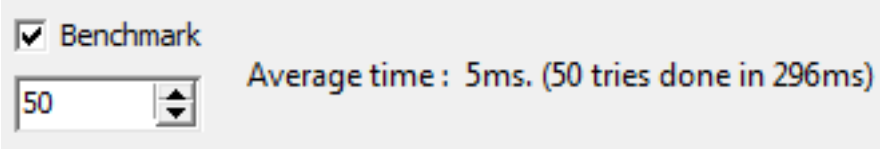


FIG. 7.4 – Résultat d'une recherche négative sur une carte de 75*75 cases

### 7.1.3 Calculer un chemin dans le jeu

Depuis la première soutenance et la mise en place du serveur SVN, nous avons vraiment plus facilement intégrer notre travail à un fichier de projet commun. Ce dernier m'a notamment permi de bien comprendre et d'intervenir dans la gestion des données, dans tout le système d'ordres à une unité par exemple, puisque l'ordre de déplacement n'est pas le seul à demander une recherche de chemin comme nous l'avons vu plus tôt. L'intégration dans le projet n'a pas été sans problèmes, mais finalement je suis assez satisfait de la recherche de chemin en elle-même puisqu'elle se comporte comme prévu, c'est à dire qu'elle ne fait pas ralentir le jeu, même avec plusieurs demandes de chemin à la fois.

### 7.1.4 Les améliorations que je souhaiterais apporter

Bien que pour le moment le besoin ne se fasse pas sentir, il semble évident qu'à un stade plus avancé, particulièrement avec d'éventuelles animations et la gestion d'un plus grand nombre d'unités, quelques améliorations seront nécessaires concernant les performances de la rechercher de chemin. Une amélioration majeure serait une division du nombre de calculs de chemin en utilisant une structure de carte différente. La carte est pour l'instant représentée par des cases, ce qui n'est pas précis et qui en plus nécessite autant de cases que de noeuds pour l'algorithme de calcul de chemin. L'idéal serait probablement d'avoir à disposition une représentation plus générale comprennant des grandes zones de déplacement liées entre elles, permettant d'avoir une idée du chemin à prendre. Dans la pratique, ce genre découpage permet par exemple dans un hypothétique voyage cherchant à relier Berlin et Rome à vol d'oiseau, à voir qu'il faut d'abord rejoindre l'Autriche, puis rejoindre l'Italie.

FIG. 7.5 – Pour rejoindre le point vert et le point bleu, on passera par les zones rosées

## 7.2 Déplacements d'unités

### 7.2.1 Donner les ordres

En plus de leur trouver un chemin, mon rôle consiste à déplacer les unités, et particulièrement à gérer un groupe d'unités. En étroite collaboration avec le responsable de gestion des données et le responsable moteur 3D et interface, nous avons mis en place un système permettant de sélectionner les unités à l'écran, de gérer cette selection, et de leur donner des ordres. Le système de gestion d'ordres à une unité a permi de facilement passer d'un mouvement rectiligne sans tenir compte des obstacles, à un appel à la fonction de recherche de chemin.

Il est actuellement possible grâce à ce système de :

- sélectionner une ou plusieurs unités alliées.
- sélectionner une unité ennemie.
- donner à toutes les unités sélectionnées un des ordres suivants :
- donner l'ordre d'attaquer un ennemi
- donner l'ordre de se déplacer
- donner l'ordre de réparer/construire un bâtiment
- donner l'ordre de suive un allié

### 7.2.2 Les améliorations attendues

Le système de déplacement, avec la recherche de chemin, permettent actuellement de donner des ordres simples, à un nombre très restreint d'unités à la fois. Les possibilités sont assez réduites, et rapidement il faudra permettre de donner des ordres complexes à plusieurs unités à la fois, tout en évitant les ralentissements. Un exemple d'optimisation sera par exemple de gérer le déplacement en groupe plutôt que donner à chaque unité sélectionnée le même ordre de déplacement, et donc à chacune un calcul de chemin. Il faudra vraiment considérer le groupe comme une entité et essayer le moins possible de le disperser. D'autres améliorations sont à prévoir, notamment le fait de différencier bâtiments et unités. En effet, pour le moment, une unité calculant un nouveau chemin comptera comme obstacle n'importe quelle autre unité. Le problème est qu'elle considère après son chemin comme fiable, alors que la position des unités sur la carte n'est pas garantie, elles peuvent avoir bougé après le calcul, et donc au mieux occasionner un détour inutile, et au pire entraîner une collision entre les 2 models 3D, ce que nous essayons d'éviter un maximum.

# Chapitre 8

# *Conclusion*

Tout s'est encore parfaitement déroulé entre nous lors de cette préparation de soutenance, l'ambiance est au beau fixe malgré les nombreux problèmes rencontrés lors de la mise en commun de nos travaux. Nos efforts ont étés récompensés puisque nous sommes satisfait du travail accompli avant cette soutenance intermédiaire et vu que l'étape de mise en commun a été accomplie, nous espérons que désormais nous pourrons nous focaliser sur l'amélioration du contenu et non plus sur l'adaptation de notre travail aux parties des autres. Nous n'allons pas nous arrêter en si bon chemin et nous allons perséverer afin d'aboutir sur un jeu fini, jouable et intuitif.