Project : HGF

Rapport de soutenance

première soutenance, le 9 janvier 2008

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

Table des matières

Chapitre 1

Introduction

Trois mois après la formation du groupe "Fooo Team", il n'a jamais été aussi soudé. En effet la préparation de cette soutenance s'est effectuée dans la bonne humeur avec une quantité de travail fourni importante.

1.1 Real Time Strategy

Nous sommes en train de réaliser un RTS (Real Time Strategy) où le joueur va être amené à construire une base, et une armée en devant gérer ses ressources afin de battre son ennemi qui aura les mêmes chances que lui.

Afin de ne pas perdre de temps dans une phase de reflexion trop longue nous allons nous baser sur le célèbre jeu Warcraft III (réalisé par Blizzard). Notre but est d'apprendre à coder un jeu de A à Z. Si nous voulons à la fin obtenir un résultat jouable et amusant il faut raccourcir certaines étapes de développement. Nous avons donc fait le choix de réutiliser les éléments graphiques et concepts de ce jeu à grand succès.

Cependant nous voulons éviter d'utiliser des fichiers appartenant à Blizzard lors de la phase finale du jeu. Il faut savoir que la communauté de moddeurs du jeu est très importante ainsi il est possible d'utiliser leur travail avec leur autorisation pour un résultat de bonne qualité. D'autant plus qu'aucun d'entre nous ne possède de qualités artistiques suffisantes. L'interface actuelle réalisée par Unwirklich (c.f. annexe) en est un bon exemple.

1.2 OpenGL

Comme sûrement tous les groupes, le choix entre OpenGL et DirectX a demandé réflexion. Au départ penchant vers DirectX nous nous sommes finalement tournés vers OpenGL non pas parcequ'il est plus simple ou plus adaptés car les deux moteurs parviennent au même résultat mais surtout pour pouvoir travailler à l'école. En effet DirectX n'est pas compatible avec les machines du parc informatique d'EPITA. Par exemple le jeu Warcraft III est jouable entièrement sous OpenGL aussi que sous DirectX.

1.3 Communication

Outre le fait que nous nous sommes beaucoup attaché à produire un code propre, qui fonctionne avec un rendu agréable à utiliser, nous avons passé du temps sur des éléments de communication qui rendent le jeu beaucoup plus attractif.

Chapitre 2

Réseau et communication

2.1 Résultat actuel

Comme convenu dans le cahier des charges, je m'occupe de la conception du site web du projet, de la réalisation d'un réseau permettant à au moins deux personnes de s'affronter dans le jeu, ainsi que de l'aspect « marketing » du projet.

2.1.1 Le marketing

Pour cette partie cela s'est résumé en la diffusion de fond d'écrans à l'effigie de la Fooo Team sur le site web, en la réalisation d'une vidéo d'introduction mettant en valeur notre équipe, et la conception de chemises personnalisées affichant fièrement le fameux logo. L'égocentrisme est de rigueur pour se démarquer dans un marché de projets de fin d'année en constante progression.

Les fonds d'écrans ont été réalisés avec le célèbre Photoshop d'Adobe, et sont donc disponibles dans la section téléchargements du site web officiel de la Fooo Team.

La vidéo d'introduction a été réalisée avec le logiciel Vegas 7 de Sony. Pour le logo animé, l'idée était de partir du symbole « biohazard » pour le transformer en les trois « o » du mot Fooo. Le plan de la conception de ce logo animé est présenté en annexe à la fin de ce document.

Il nous était tout d'abord venu l'idée d'uniformiser la tenue de la Fooo Team lors des soutenances grâce à des tee-shirts exhibant le logo de notre équipe ; la possibilité de mettre des logos sur des chemises nous a paru séduite pour son côté plus professionnel. Il m'a donc fallu trouver une entreprise qui permettait d'imprimer notre logo sur un vêtement que nous fournissions.

C'est le Studio Chaillot[1] que j'ai retenu pour réaliser cette tâche. Après une après-midi de shopping rendue possible grâce à une absence de M. Digni, et une petite heure d'impression, nous voilà donc habillés pour nos représentations.

2.1.2 Le site web

La réalisation du site web fut pour moi une expérience très enrichissante. Je n'en étais pas à mes premiers pas en matière de réalisation de sites web, mais je n'avais jamais vraiment réalisé le mien de A à Z. Mon but était de réaliser un site à la fois simple et « propre » au niveau de sa conception. Le respect d'une sémantique et des standards du web sont restés pour moi une priorité.

Le développement de ce site m'a fait découvrir différents langages : le xHTML pour la sémantique, le CSS pour la mise en forme, le PHP pour la structure dynamique des différentes pages et localisations, et enfin le XML pour stocker le texte brut, selon la langue désirée.

Ce site sera en effet localisé selon la langue de l'utilisateur : un fichier XML différent sera appelé en fonction de la langue choisie. Il me parait essentiel de disposer d'une version anglaise du site web afin d'élargir sensiblement le public visé.

Le site web de la Fooo Team est disponible à l'adresse : www.fooo-team.com

2.1.3 Le réseau

Après avoir réalisé la majeure partie du site web, je me suis consacré à ma tâche principale dans ce projet : la réalisation d'un système de réseau permettant à plusieurs joueurs de s'affronter dans des parties homériques.

L'idée est tout d'abord de créer un système de serveur, auquel pourrait se connecter un nombre indéterminé de clients, échangeant de simples chaînes de caractères. Ce qui pourrait s'apparenter à un simple « chat » se transformera, une fois ce texte interprété comme des commandes de jeu, en un centre de synchronisation des différents joueurs, en clair, un moteur réseau.

J'ai donc commencé à utiliser l'outil « TForm » de Delphi pour faire deux fenêtres simples, un serveur et un client, avec chacune une boîte de texte qui affiche les dernières actions effectuées.

Pour le réseau en lui-même, j'avais la connaissance de l'existence de plusieurs outils : Les outils « internet » de Delphi, très basiques, le composant

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

Indy, lui aussi intégré à Delphi, et la librairie ICS, externe à Delphi. J'ai rapidement éliminé cette dernière, puisqu'il m'a été impossible de l'installer sur les machines du parc informatique de l'EPITA ; voulant pouvoir à la fois travailler chez moi, et dans les locaux de l'école. J'ai commencé alors à travailler avec les outils « internet », se résumant à deux composants : « Ttcpserver » et « Ttcpclient ». Je me suis très rapidement confronté aux limites de ces outils, autant au niveau de leur puissance que par leur peu de documentation et d'expériences d'utilisation sur internet. J'ai donc utilisé le composant Indy, qui est un bon compromis entre les deux : intégré à Delphi et complet.

J'ai donc pour le moment établi une connexion entre un (ou plusieurs) client(s) et un serveur, et échangé des données sous forme de chaîne de caractères, qui se contentent pour l'instant d'être affichées dans la boite de texte.

2.2 Ce qu'il reste à faire

2.2.1 Le marketing

Le gros de la partie marketing sera la réalisation de la jaquette, du livret d'utilisation du jeu et de tout ce qui tournerait autour d'une commercialisation du produit. Cela ne pouvant se faire qu'au dernier moment, mon rôle dans le marketing se décidera au fur et à mesure des nouvelles idées du groupe pour conquérir le marché : référencement approfondi du site web, bande annonce du jeu, réseau de publicité à l'intérieur des groupes de projets de SUP, etc.

2.2.2 Le site web

La structure principale du site web ne subira probablement plus de modifications majeures. Les améliorations apportées seront donc désormais de l'ajout de contenu ou de nouvelles traductions. En effet, comme mentionné précédemment, le site est déjà conçu pour accueillir un nombre indéterminé de versions différentes, la traduction ne nécessite maintenant plus qu'un travail linguistique.

2.2.3 Le réseau

C'est maintenant le réseau qui me prendra le plus de temps. Je n'ai pas encore d'idées très précises du déroulement de son développement : il me faudra optimiser au maximum les dialogues entre les différentes parties de ce réseau afin de pouvoir transmettre un grand nombre de données, et ce dans des intervalles de temps très courts. Il me faudra ensuite travailler en étroite collaboration avec Félix afin de traduire ces données en commandes permettant de diriger les unités du jeu.

Chapitre 3

Gestion de données

3.1 Choix du XML

Nous avons décidé de stocker les données du jeu sous le format XML dont voici les avantages.

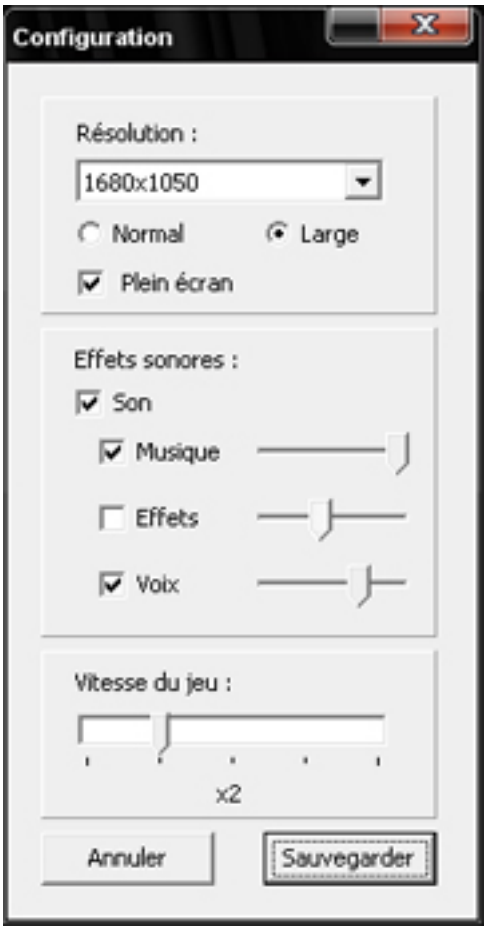
Les fichiers sont facilement manipulables, lisibles et compréhensibles sous leur forme brute. Le XML est un standard universel et Open Source recommandé par le W3C et pouvant représenter tous les types usuels de données.

Il présente une structure arborescente, c'est-à-dire que les informations sont contenues dans des noeuds étiquetés et les noeuds sont en relation par le biais de liens de parenté père-fils. Il est donc nécessaire de récupérer les données contenues dans les noeuds à l'aide d'un algorithme de parcours de XML, fort heureusement il existe de nombreux analyseurs fonctionnels de manipulation comme le Document Object Model (DOM) qui a été utilisé sous Delphi pour lire les fichiers du jeu.

3.2 Les Options

Pour m'initier à la gestion de données, j'ai commencé par coder un outil de modification des options du jeu. Les préférences spécifiées par l'utilisateur sont chargées au lancement depuis un fichier XML et mémorisées dans un enregistrement.

Ces préférences sont modifiables en réglant différentes options comme le mode plein écran, le volume de la musique, des voix ou des effets sonores ou encore le choix de la résolution (normale ou large) parmi une liste générée en fonction des résolutions possibles pour l'ordinateur. Puis les modifications peuvent être sauvegardées en réécrivant sur le fichier d'options. Enfin, pour protéger de toutes modifications accidentelles du fichier, ou si il est absent, la résolution actuelle de l'écran est utilisée et les autres options reçoivent des valeurs par défaut.



3.3 Editeur d'unités et de compétences

J'ai ensuite développé un éditeur d'unités et de compétences pour en faciliter l'ajout dans le jeu. Encore une fois, tout est stocké sous format XML dans deux fichiers séparés :

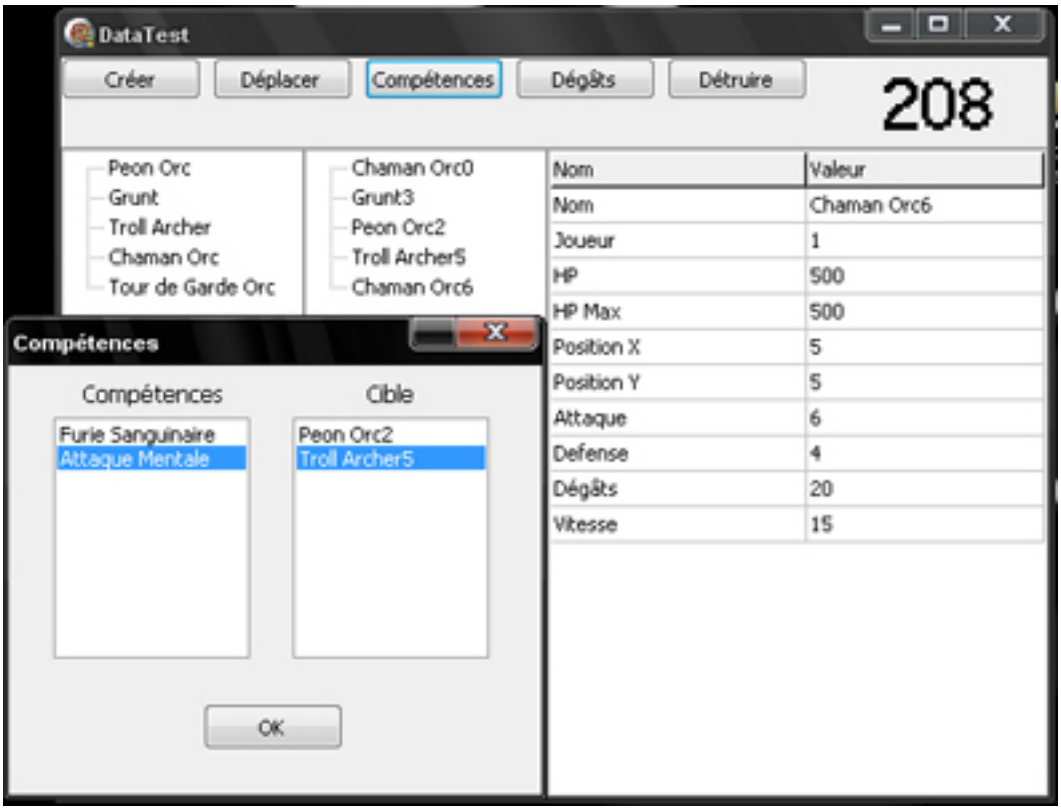
- Dans les fichiers unités, différentes caractéristiques sont stockées : son nom, ses points de vie, ses compétences, ses dégâts, sa défense, sa vitesse de déplacement, si c'est un bâtiment, etc.
- Dans les fichiers compétences : nom, durée, cibles possibles, effets sur la cible.

Il est ensuite possible de modifier chaque attribut, de rajouter une compétence à une unité, d'en supprimer une, de rajouter des bâtiments pouvant être construits par cette unité, etc. Il est aussi possible de créer ou de supprimer des unités ou des compétences. Puis, toutes les modifications peuvent être sauvegardées dans les deux fichiers XML afin d'êtres chargées au lancement du jeu.

3.4 Simulation des données du jeu

Les données des unités et des compétences sont encore une fois chargées depuis les mêmes fichiers XML. Il s'agit de simuler la gestion des données dans le jeu (à l'aide d'une TForm pour l'instant). Il est possible de créer une unité parmi la liste chargée depuis le fichier et de la placer à une certaine position.

On peut ensuite déplacer les unités sur une case adjacente, utiliser une compétence si une unité est à portée. Les effets sur la cible seront actifs pendant une durée égale à celle de la compétence (en secondes). Enfin, si une unité subit assez de dégâts, elle est détruite.



3.5 Ce qu'il reste à faire

Il faut désormais continuer la simulation de gestion de données en empêchant par exemple que plusieurs effets identiques s'accumulent sur une même unité et en gérant de manière plus efficace la durée des compétences.

Puis, il faudrait commencer à mettre en commun le travail pour que la gestion des données soit visible via le moteur graphique et que les données soit echangées en réseau pour permettre de jouer en ligne. Enfin, il sera nécessaire de travailler l'optimisation afin de réduire au maximum les calculs effectués à chaque tour de jeu.

Chapitre 4

Déplacements d'unités

4.1 Mon rôle dans l'équipe

4.1.1 Rôle de chef

Je suis le chef de projet de la Fooo Team, ce qui n'est en fin de compte pas si peu. Notre projet étant assez ambitieux, nous avons été confrontés de nombreuses fois à des choix pour lesquels j'ai souvent dû trancher.

Au final, ne pas revenir sur les décisions a été bénéfique en nous permettant de moins longtemps hésiter sur celles-ci, et donc de passer plus rapidement à d'autres problème, voire tout simplement (éventuellement) à la suite.

4.1.2 Des déplacements plausibles

Ma fonction principale est de permettre au final d'obtenir le déplacement "intelligent" des unités d'un joueur sur la carte tout en respectant des règles de physique basique :

- Les unités ne doivent pas pouvoir traverser certains obstacles.
- Deux unités ne doivent pas se trouver au même endroit ou se recouvrir.
- Quand une unité reçoit l'ordre de se déplacer à un endroit il faut qu'elle trouve "elle-même" un chemin jusqu'à cet endroit avec le moins de détours possible.
- Quand plusieurs unités reçoivent un même ordre simultanément, elles doivent former un groupe, et essayer de lui rester fidèle. Il faudra donc éviter que des unités d'un même groupe n'empruntent des chemins différents, et donc logiquement trouver un chemin unique pour le groupe.

4.2 Recherche d'un chemin

4.2.1 Le pathfinding

Je me suis d'abord concentré sur ce qui m'a paru le plus urgent, comprendre et implémenter, un algorithme de recherche de chemin le plus court (plus communément appelé algorithme de pathfinding).

En effet, c'est avec un algorithme de ce type que les unités pourront non seulement chercher un chemin mais aussi éventuellement si elles rencontrent des obstacles imprévus, recalculer leur chemin. Il m'a donc semblé important de pouvoir produire une fonction générique de recherche de chemin

Après quelques recherches sur le sujet, mon choix s'est rapidement porté sur l'algorithme de l'A* (a star), qui m'a semblé suffisamment simple au début et qui après plusieurs optimisations semble un des algorithmes de pathfinding basiques les plus satisfaisant au niveau des performances.

Globalement, cet algorithme permet de trouver un chemin la plupart du temps parmi les meilleurs chemin possibles, mais par rapport à d'autres algorithmes qui trouvent toujours le meilleur chemin, il est très rapide.

4.2.2 Principe de fonctionnement de l'A*

Les bases de l'algorithme

On appellera node chaque point qu'il est possible d'atteindre dans le *monde* dans lequel on recherche un chemin. L'algorithme de l'A* repose sur 2 notions propres à chaque node : leur *poids* et leur *parent*, ainsi que sur 2 listes de nodes que nous appelerons *liste ouverte* et *liste fermée*.

La liste ouverte contiendra toutes les nodes que l'on doit traiter.

La liste fermée contient toutes celles qui ont déjà été traitées.

Le poids d'une node est donné par la somme de la distance qu'on a parcouru depuis le départ et d'une estimation de la distance restante jusqu'au point d'arrivée. Dans le cas d'un terrain sous forme de damiers par exemple, on calcule estime cette distance avec le calcul de la distance dite de Manhattan, c'est à dire la somme de la différence des X et des Y.

Le parent d'une node est tout simplement un moyen de savoir quelle autre node l'a précédée.

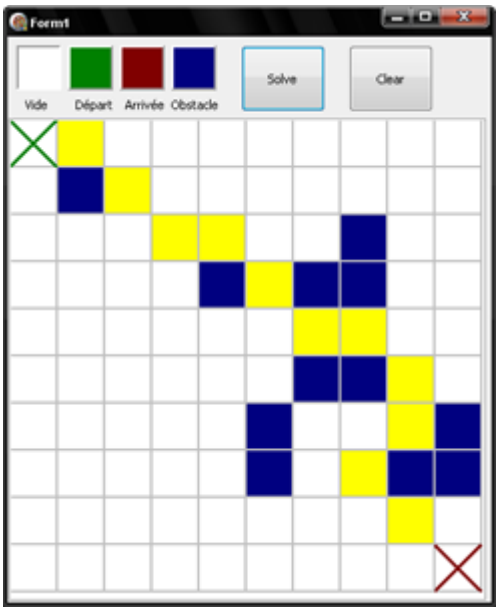
L'algorithme en plein travail

```
Ajouter node_depart à liste_ouverte

Tant que liste_ouverte non_vide répéter :
| node_actuelle <- node au poids minimum de liste_ouverte
| retirer node_actuelle de liste_ouverte
| ajouter node_actuelle à liste_ferme
| Pour chaque node que node_actuelle peut atteindre et pas dans liste_fermée :
| | Si elle est dans liste_ouverte
| |   Si distance(node/depart) < distance(node_actuelle.parent/depart)
| |     node_actuelle.parent <- node
| |   Sinon
| |     ajouter node à liste_ouverte
  /
/
```

4.3 Simulation d'un trajet

Pour pouvoir facilement développer cet algorithme avant d'avoir l'interface définitive du jeu, j'ai utilisé les Tforms que Delphi met à notre disposition. Les éléments TstringGrid par exemple, même s'ils ne sont pas forcément intuitifs pour un débutant, sont vraiment adaptés à la réalisation d'une petite interface pour simuler un départ, une arrivée et des obstacles.



4.4 Ce qu'il reste à faire

Le travail qu'il me reste à accomplir est globalement la partie concernant les collisions entre unités et les déplacements de groupes d'unités. Avant toute chose nous allons probablement très bientôt mettre en commun nos travaux pour avoir plus rapidement une idée des points les plus critiques.

Chapitre 5

Moteur graphique et interface

Etant au centre du projet, j'ai relativement bien avancé pour que le travail des autres membres du projet puisse s'intégrer lors de la seconde soutenance. Les bases sont posées, les prochaines étapes de développement vont constituer à les consolider et à construire par-dessus.

5.1 Partie Interface

5.1.1 Introduction

De trop nombreux jeux (commerciaux ou non) négligent l'interface utilisateur, l'envisagent qu'en dernier dans le processus de développement. On arrive à la fin à un jeu qui n'est pas du tout intuitif, et pas amusant alors qu'il a toutes les qualités nécessaires pour l'être.

Pour ne pas refaire les mêmes erreurs, et grâce à mes connaissances tirées de la modification de l'interface de World of Warcraft nous avons décidé de s'en occuper dans les premières étapes de développement.

Pour que l'on puisse travailler sur celle-ci plus facilement il nous faut un moyen de pouvoir la scripter sans pour autant qu'entre chaque modification il faille recompiler l'exécutable et relancer le jeu ce qui prend un temps non négligeable.

World of Warcraft utilise le Lua comme langage de programmation pour son interface. Etant donné que celui-ci possède des liens qui fonctionnent sous Delphi il parut être la meilleure solution.

5.1.2 Intégration

L'intégration de celui-ci se relève assez simple. Afin de démarrer il nous suffit de savoir faire trois choses :

- Charger un fichier Lua
- Appeler une fonction écrite en Lua à partir de Delphi en lui donnant des paramètres et en récupérant les résultats
- Appeler une fonction écrite en Delphi à partir du Lua en lui donnant des paramètres et en récupérant le résultat

La communication entre les deux langages s'effectue grâce à une pile. En effet pour appeler une fonction Lua il faut ajouter au sommet de la pile tous les paramètres un à un puis appeler cette fonction. A la suite de l'exécution au sommet de la pile se trouvent toutes les valeurs retournées (Le Lua n'est pas limité à une valeur retournée) ou alors un message d'erreur qu'il faut traiter.

Pour l'instant les seules fonctions Delphi que nous avons donné au Lua sont

- Print : Ecrit un message dans le fichier de Log
- SendChatMessage : Envoie un message par le biais du réseau
- IncludeFile : Charge un fichier Lua
- RunScript : Execute une chaine Lua
- ReloadUI : Recharge l'interface

De nombreuses autres fonctions seront ajoutées une fois que le jeu sera plus avancé. Celles-ci permettent de commencer la programmation de l'interface. Nous aborderons la partie graphique de l'interface plus tard dans le document.

5.2 Partie Moteur Graphique

5.2.1 Introduction

A l'instar de la majorité des groupes, la première chose sur laquelle nous nous sommes penchés n'était pas l'affichage d'un cube mais le chargement d'un modèle. Etant donné que notre but n'est pas de concevoir des modèles et afin d'avoir un jeu beau graphiquement nous avons décidé d'utiliser les modèles de Warcraft III.

Ce sont des fichiers binaires d'extension .mdx, j'ai donc cherché comment les afficher à l'écran. Tout d'abord il faut comprendre comment lire le fichier. J'ai donc fait appel à des contacts qui avaient déjà travaillé dessus. Ils m'ont fourni de la documentation sur le fichier en lui-même et des sources de plusieurs « Model Viewer ».

5.2.2 Analyse du fichier

La structure des modèles est relativement simple, c'est une succession de blocs et de sous-blocs dont on connait la taille et la celle des sous éléments. La seule donnée manquante est le nombre d'éléments que contient chaque bloc. Grâce à un système de compteur on arrive à lire correctement le fichier.

La première étape consiste à construire le « moule » dans lequel on va glisser ce que l'on va lire. Chaque sorte de donnée va être représentée sous la forme d'un enregistrement. C'est-à-dire une unité structurée de données qui peut contenir des données de types différents. Par exemple voici la représentation d'un Noeud.

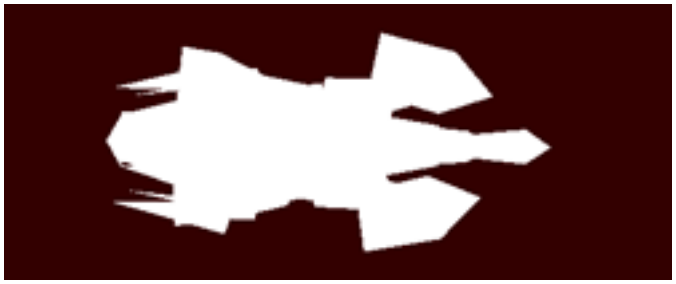
```
NodeChunk = Record
  Name            : string[80];
  ObjectId        : Dword;
  ParentId        : Dword;
  Flags           : Dword;
  NodeTranslation : Transformation3;
  NodeRotation    : Transformation4;
  NodeScaling     : Transformation3;
end;
```

Une fois que la structure du modèle a été transcrite en Delphi il faut alors la remplir. Etant donné que la structure n'est pas fixe, c'est-à-dire que des éléments peuvent ou ne pas être présents et que l'on ne connait pas le nombre d'élément dans chaque bloc, il n'est pas possible de réaliser un analyseur généraliste qui remplit la structure toute seule.

Pour faciliter cette tâche nous avons découpé le problème en sous-fonctions. Chaque bloc, sous-bloc et valeur simple est défini par une fonction. Il suffit alors d'appeler consécutivement ces fonctions suivant la documentation de la structure. Il faut juste faire attention de garder un compteur afin de ne pas empiéter sur le bloc suivant.

5.2.3 Affichage du modèle

Avoir le modèle chargé en mémoire n'est pas suffisant, il faut maintenant pouvoir l'afficher à l'écran. C'est maintenant que nous allons avoir besoin des fonctions OpenGL. Une fois l'initialisation terminée, il suffit à chaque fois que l'interface est dessinée d'afficher tous les triangles du modèle.



Un modèle de Zergling réalisé par un français Abriko a été utilisé dans les exemples car il est relativement simple, texturé, animé et nous avons l'autorisation de l'utiliser.

Un modèle tout blanc n'est pas très utile c'est pourquoi il faut lui ajouter une texture. Celles de Warcraft III sont des fichiers BLP, un format propriétaire qui a pour base du DXT (DirectX Texture). Afin de ne pas avoir à coder une fonction de chargement de texture nous avons converti toutes les textures en TGA et utilisé celle réalisée par Jan Horn (Sulaco) (Demande d'autorisation en cours).

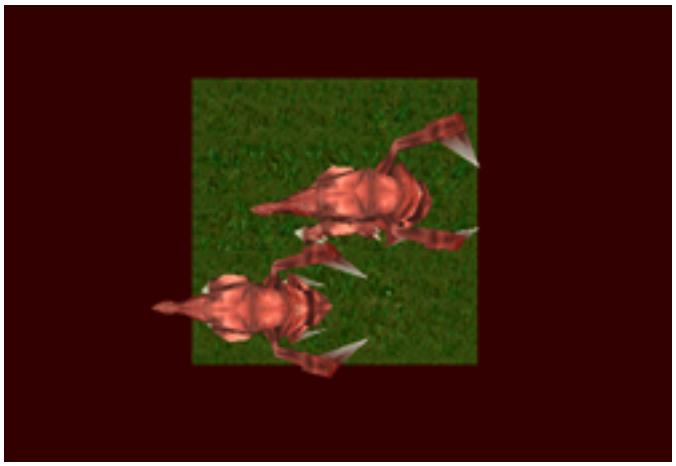
Encore une fois les modèles étant très bien fait l'application de la texture est très simple à mettre en place.

5.2.4 Affichage du terrain

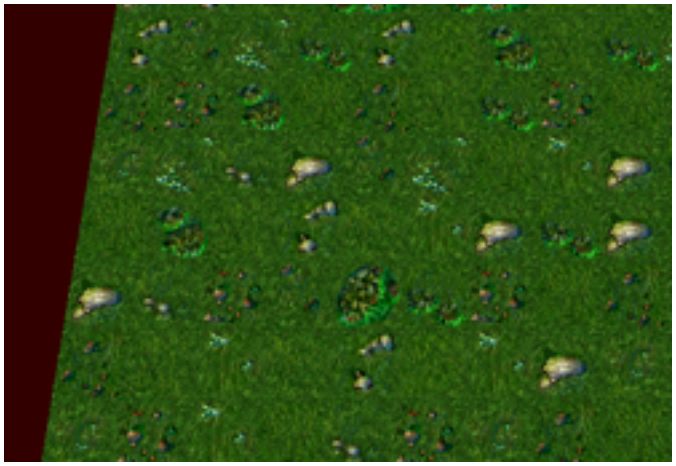
Notre zergling est bien beau mais il faut un terrain sur lequel il puisse reposer. La première chose à faire est d'afficher un carré qui va servir de base.

Après quelques problèmes de transparences corrigés on applique une texture au terrain.

Les pauvres petites bêtes se sentent seules sur leur minuscule îlot. Maintenant que nous avons réussi à charger un carré de texture, il va falloir en aligner beaucoup plus pour former une vraie carte.

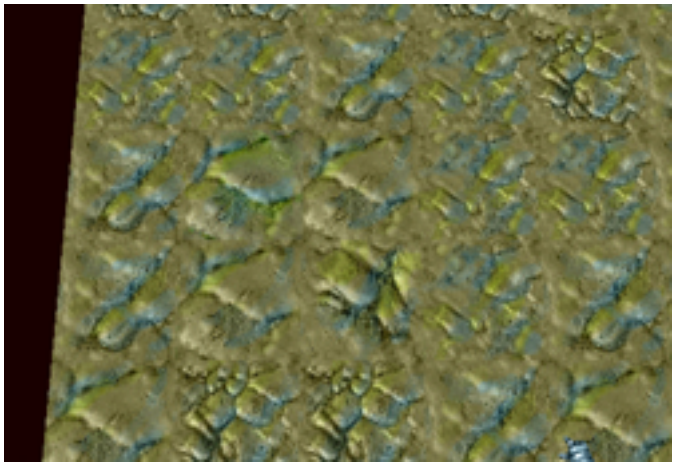


Pour la rendre plus réaliste, nous allons placer aléatoirement une des nombreuses textures dont nous disposons pour chaque type de terrain.



Il faut cependant raffiner ce tirage aléatoire afin d'obtenir un résultat convenable.

On peut également penser être dans une carte qui se situe en montagne donc on peut charger des textures de sol rocheux.



Disposant d'une carte nous nous sommes penchés sur d'autres aspects du jeu plus essentiels. Cependant il serait important dans le futur de permettre d'afficher des textures d'herbe et de sol côte à côte avec une transition.

5.2.5 Déplacement de la Caméra

Il est bien beau d'avoir une carte énorme mais encore faut-il pouvoir s'y déplacer. Après réflexion nous allons donner à l'utilisateur la possibilité de se déplacer selon 3 méthodes

– Les flèches directionnelles

– La souris sur le bord de l'écran
– Un « drag » de la souris sur la carte avec le bouton central appuyé

Ces trois méthodes ne sont pas en elles mêmes compliquées mais nécessitent des composants qui n'ont pas encore été codé.

Tout d'abord il a fallu réaliser le module de gestion des entrées claviers. Celui-ci a tout de suite été pensé de façon à pouvoir changer les touches dans le futur.

Ensuite une gestion du curseur a été faite. Alors que nous étions en train de fonctionner dans un environnement en trois dimensions il a été nécessaire de comprendre comment passer dans le repère à deux dimensions que définit l'écran. Ensuite on place un carré texturé à partir d'une image contenant de nombreux curseurs à la position de la souris.

La où cela se complique c'est lorsque la souris est sur un bord de l'écran, celle-ci se transforme en un triangle indiquant la direction et le sens du déplacement de la caméra. L'image ne contient qu'une seule flèche donc il faut la tourner dans le bon sens. Ensuite la position d'application de la texture est différente selon chaque cas. La pointe de la flèche doit être à la position du curseur.

5.2.6 Interface

Une fois la gestion du curseur finie, il fut très simple d'afficher l'interface réalisée par Unwirklich. C'est purement dans un but esthétique que nous la faisons apparaître à ce stade du développement. Afin que l'interface soit simple à coder nous avons décidé de la coder dans des fichiers XML qui seront analysés. Vous pouvez voir l'interface actuelle en annexe.

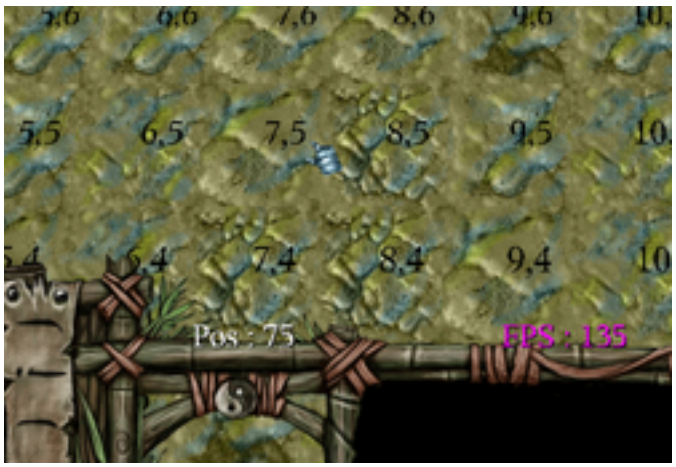
5.2.7 Texte

Lorsque l'on code, il est souvent nécessaire de pouvoir écrire du texte quelque part pour tester des éléments précis. On a donc développé un petit module qui permet de rajouter une ligne dans le fichier « Logs/log.txt » (communément appelé fichier de log) avec l'heure exacte et le degré de gravité du message. Un message avec un degré de gravité trop important peut interrompre l'exécution du programme.

Cependant il se révèle beaucoup plus pratique de pouvoir afficher du texte directement à l'écran. D'après les informations recueillies il existe deux méthodes bien distinctes pour afficher du texte.

On peut tout d'abord à partir d'une police générer une image de tous les caractères et ensuite les afficher comme des carrés de texture. Celle-ci a le mérite de permettre des effets sur le texte facilement mais requiert de générer une image pour chaque police, ce qui n'est pas forcément très pratique.

L'autre solution pour laquelle nous avons opté est à partir d'une police de générer un polygone de chaque caractère et le stocker dans une liste. Il suffit ensuite d'appeler cette liste pour afficher une chaine de caractère. C'est une solution qui marche sur toutes les polices sans avoir besoin de générer une image, elle permet à la fois d'afficher du texte en 2D sur l'interface mais aussi du texte en 3D.



5.3 Ce qu'il reste à faire

Les bases du jeu sont posées, il faut maintenant les consolider. Au sein de la partie interface tout est à faire pratiquement. Le premier pas sera d'avoir un analyseur de fichiers XML qui permette d'afficher les parties de l'interface correctement et qui puisse être modifié par le Lua au cours du jeu. Une fois celui-ci plus ou moins stable il faut absolument avoir une boîte de texte qui fonctionne pour continuer le développement du jeu.

Pour ce qui est de la partie graphique, le chargeur de modèle en est encore à ses balbutiements. Ils sont affichés relativement correctement mais une grosse optimisation est à revoir car deux bâtiments affichés suffisent à réduire le FPS considérablement. De plus les animations ne fonctionnent pas et tout un tas d'options telles que la gestion des lumières, les générateurs de particules, la gestion de la couleur d'équipe ...

Maintenant que la base est posée, il faut également tout centraliser : le pathfinding, les options et la gestion des unités ainsi que le moteur réseau doivent être réunis.

Chapitre 6

Conclusion

C'est avec beaucoup d'enthousiasme que nous avons travaillé pour ce projet, qui est pour nous quatre passionnant et motivant.

Nous allons tenter par la suite de tenir ce rythme de développement autant au niveau de la conception du jeu en lui-même qu'à celui des produits dérivés et autres produits publicitaires tournant autour de ce projet.

Enfin, et c'est l'essentiel, nous espérons tout simplement avoir pu donner l'envie au lecteur de jouer au jeu final, et nous l'invitons à patienter jusqu'à juin.

Chapitre 7

Annexe

FIG. 7.1 – Croquis de réalisation de la vidéo

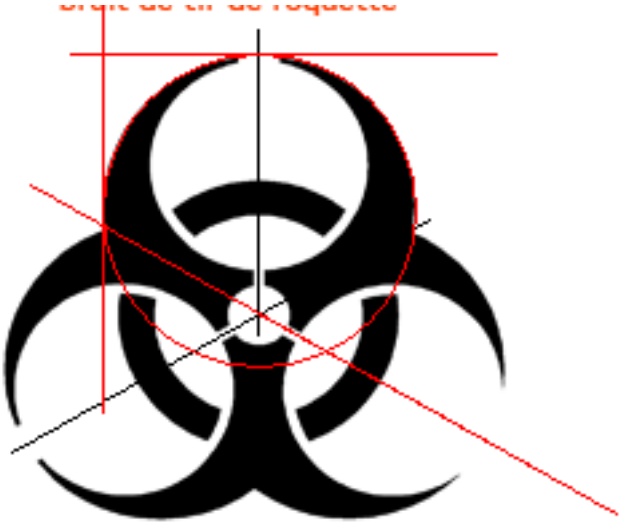


FIG. 7.2 – Patron de rotation du symbole

FIG. 7.3 – Interface réalisée par Unwirklich