DA GNIII PROJECT

RAPPORT DE PROJET

Table des matières

I Introduction **4**

II Cahier des charges **5**
1 Avant-propos 5
2 Introduction 5
3 DGP 6
4 Objet de l'étude 7
5 Découpage du projet 7
5.1 Moteur graphique 7
5.2 Moteur physique 7
5.3 Sons et graphismes 8
5.4 Réseau 8
5.5 Gameplay 8
5.6 Site web 8
5.7 Finalisation du projet 8
6 Répartition du travail 9
6.1 Première soutenance 9
6.2 Seconde soutenance 9
6.3 Troisième soutenance 10
6.4 Soutenance finale 10
7 Moyens 11
8 Conclusion 11

III Réalisation **12**
1 Graphismes 12
1.1 Affichage des maps 12
1.2 Loader 3DS 14
2 Physique 16
2.1 Octree 16
2.2 Collisions 19
2.3 Forces 23
3 Le son 25
4 Gameplay 28
4.1 HUD 28
4.2 Menus 30
4.3 Contrôles 33
4.4 Effecteur 35
4.5 Gestion d'effets 37
5 Reseau 40

6 Divers 43
6.1 Système de classes 43
6.2 Threads 45
6.3 Parseur de fichiers 48
7 Site web 51
8 Finalisation 55

IV Récit de la réalisation **57**

V Impressions **60**
1 Laurent 60
2 Florent 61
3 Renaud 62

VI Conclusion **63**

I Introduction

Mardi 13 juin, 2h15 : le projet touche à sa fin. Plus que quelques détails de dernière minute à régler et le rapport de projet à rédiger. Dehors il fait chaud, très chaud et je repense à toutes ces heures passées à coder le projet, rédiger les rapports, et soutenir l'insoutenable.

D'ici quelques heures DGP ne sera plus qu'un souvenir. Et pourtant quel souvenir ! Comment résumer en une cinquantaine de pages ce qui a animé nos vies presque au quotidien ? Et pourtant c'est le but de ce dernier rapport.

Nous commencerons donc par une reprise du cahier des charges, quelque peu modifiée par rapport à la version originale. Nous continuerons ensuite notre exposé par la présentation, domaine par domaine, des divers éléments composant notre projet avant de conclure sur le récit de la réalisation et nos impressions personnelles.

Bonne lecture, et à bientôt pour de nouvelles aventures !

II Cahier des charges

1 Avant-propos

Cette version du cahier des charges n'est pas la version originale, nous vous proposons une version adaptée, prenant notamment en compte les changements dans le groupe (pour rappel Nicolas Aycardi a arrêté, et Renaud Durand a remplacé Alexandre Bénière).

2 Introduction

Né à Paris en Juin 1905, Le petit Jean-Paul fut élevé par sa mère Anne Marie Schweizer. Après une enfance pas très cool, il rencontra un copain, avec qui il fit des trucs assez sympatoches. Mais bon, ça n'a rien à voir avec notre projet tout ça.

Née à Paris en 2005, l'équipe Hexen, fut maternée par (et là faut ajouter les noms). Après une première année 2005-2006 difficile, passée à concevoir pour l'EPITA un projet de FPS trop cool nommé DGP, l'équipe fut reconnue internationalement et fit rapidement fortune, accumulant les milliards avec une facilité déconcertante. Mais ça, ça vient après le projet.

Concrètement, dans ce cahier des charges, on pourra lire des tonnes d'informations sur "Comment se prendre la tête avec un ordinateur ?", "Comment créer et agencer un tableau avec plein de chiffres en LaTeX ?", mais surtout, de nombreuses informations concernant notre projet.

Nous introduirons donc brièvement le projet lui même dans un premier temps, avant de se concentrer sur les aspects plus techniques que cache sa réalisation, afin de terminer sur le planning et les ressources néces-saires à son développement.

Il ne nous reste plus qu'à vous souhaiter une agréable lecture...

3 DGP

Le *Projet Qui-Grince*, ou *Da Gniii Project*, comme nous préférerons l'appeler pour des questions évidentes de présentabilité, est un FPS WYSIWYS[1]. En gros et pour faire simple, plusieurs joueurs sont enfermés dans une sorte d'arène, avec chacun un Fusil d'Assaut M141-B Partial, un Glock 9.9mm à impulsion, un Couteau XS-RazorThin, et des fraises[2]. Le but, comme peut le deviner tout individu dôoté d'au minimum de trois neurones, est d'éliminer l'adversaire qui n'en mène pas plus large, simplement en positionnant son réticule de visée à contraste renforcé sur le pixel qui bouge là-bas au fond, de cliquer, et d'observer la multitude d'effets lumineux bicolores se refléter sur le décor, alors qu'un doux rayon laser se fraie un chemin jusqu'au ventre sensuel que l'on devine à travers le lourd tissu tramé. Et *Sprotch*.

Pour faire simple, les auteurs de ce projet étant des fans de FPS, ils souhaitaient réaliser le leur. Néanmoins, il n'était pas question de copier ce qui a déjà été maintes et maintes fois réalisé, DGP a tenté d'être le plus original possible.

Ca c'est le principe. Mais pour le faire, il a fallu quelques trucs un peu plus recherchés en programmation...

[1]First Person Shooter - What You See Is What You Shoot. Pour les réfractaires à l'anglais : Le Premier Tireur De Personne - Ce Qu'Est Vous Voyez Ce Que Vous Tirez ©Google Translator 2005

[2]Parce que c'est bon, les fraises. Néanmoins le paquetage peut être amené à changer selon la mode au moment où sortira le jeu

4 Objet de l'étude

Le but du projet étant principalement d'apprendre tout en se faisant plaisir, nous avons fait de notre mieux pour le mener à son terme.

N'ayant jamais programmé sous Delphi notre but fut dans un premier temps sera de maîtriser cet outil.

Nous devions être en mesure de développer avec Delphi un moteur graphique, un moteur physique, un client et un serveur pour le réseau, ainsi que de gérer le son, les graphismes...

Pour cela nous devions nous familiariser avec des outils que nous n'avons jamais utilisés auparavant, tel Direct X, l'octree, Winsocket...

Ce projet fut aussi l'ocassion d'apprendre à travailler en équipe : prendre des décisions, répartir les tâches, faire un planning, le respecter et surtout, ne frapper personne avant la soutenance finale.

En clair, le travail s'annonçait aussi passionnant que long, il ne nous restait plus qu'à être à la hauteur de nos ambitions.

5 Découpage du projet

5.1 Moteur graphique

Le moteur graphique est une partie très importante du projet. Il permet une animation fluide et l'utilisation de différents élements graphiques tels les modèles 3DS, les sprites, les textures. Il était responsable de l'affichage du jeu lui même, mais aussi des menus.

5.2 Moteur physique

Le moteur physique, tout aussi indispensabler que le moteur graphique, s'occupe d'intégrer à l'environnement de DGP une physique performante, permettant de rendre le jeu virtuel plus réel. Il permet la gestion des collisions et des forces et s'efforce d'être optimisé afin d'obtenir les meilleures performances possibles.

5.3 Sons et graphismes

Cette section concerne l'environnement intrinséque au jeu. Les sons, les modèles et les textures utilisés devront être représentatif de l'esprit du jeu afin d'assurer au joueur une immersion totale dans l'univers déjanté que nous souhaitions lui proposer.

5.4 Réseau

Notre jeu s'articule sur un jeu en arènes, il doit donc disposer d'une interface réseau adaptée à ses besoins. Le joueur a donc l'opportunité de jouer en réseau avec d'autres joueurs.

5.5 Gameplay

Le gameplay est une partie très importante puisque son rôole est d'assurer l'interaction entre le logiciel et le joueur. Il est donc évident qu'elle ne devait pas être négligée sous peine de rebuter l'utilisateur et donc de condamner notre logiciel à ne pas être utilisé. Le gameplay s'oriente autour de différents axes tels que les menus, les options de jeu et de configuration, la jouabilité, la difficulté d'utilisation...

5.6 Site web

Le site web est un élément important et non négligeable, intervenant tant dans la promotion que dans la survie du projet. Il est par conséquent fréquemment mis à jour, présentant ainsi l'évolution du travail au travers de textes et de captures d'écran. Les visiteurs peuvent y télécharger les sources du projet, les rapports de soutenance et le cahier des charges. Des liens utiles relatifs au projet sont également mis à la disposition des internautes.

5.7 Finalisation du projet

Cette étape concerne la finalisation du projet, c'est à dire le réglage des derniers détails d'optimisation et de compatibilité avant la distribution finale du projet. Le logiciel est être en outre accompagné d'une interface d'installation et de désinstallation. Cette partie concerne aussi l'impression du manuel et des jacquettes accompagnant le CD-ROM.

6 Répartition du travail

Renaud est absent des deux premières soutenances, puisqu'il a joint l'équipe juste avant la troisième soutenance. Alexandre n'y est pas non plus mentionné puisque pas une ligne de code n'a été apportée au projet.

Légende

ø	Pas abordé
−	Commencé
+	Bien avancé
⊕	Terminé

6.1 Première soutenance

	Florent	**Laurent**
Moteur graphique		−
Moteur physique	−	
Son et graphismes		ø
Réseau	ø	
Gameplay	ø	ø
Site web		ø

6.2 Seconde soutenance

	Florent	**Laurent**
Moteur graphique		+
Moteur physique	+	
Son et graphismes		−
Réseau	ø	
Gameplay	−	ø
Site web		+

6.3 Troisième soutenance

	Renaud	Florent	Laurent
Moteur graphique			+
Moteur physique		⊕	
Son et graphismes	+		+
Réseau	−	ø	
Gameplay		+	+
Site web			⊕

6.4 Soutenance finale

	Renaud	Florent	Laurent
Moteur graphique			⊕
Moteur physique		⊕	
Son et graphismes	⊕		⊕
Réseau	⊕	⊕	
Gameplay	⊕	⊕	⊕
Site web			⊕

Tout était prêt pour la dernière soutenance, même si plus de temps nous aurait permis d'élaborer plus de maps, de modes de jeux...

7 Moyens

Produit	Nombre	Prix
Boîtiers DVD	5	1.99 €
Feuille papier adhésif	10	7.50 €

Le projet a aussi nécessité beaucoup de feuilles pour l'impression des rapports, des cartouches d'encre pour l'imprimante et l'utilisation de certains logiciels ; néanmoins ces choses n'ayant pas été achetées spécialement pour le projet, elles ne sont pas mentionnées ci-dessus.

8 Conclusion

Tous les éléments et idées relatifs au projet ont été exposés. Ce cahier des charges définit la ligne de conduite qui fut la nôtre tout au long de la réalisation du projet.

III Réalisation

1 Graphismes

1.1 Affichage des maps

Réalisation : Laurent

Période : Octobre - Juin

Prérequis :

- Connaissances en Direct X
- Parseur de fichier
- Maths (matrices, vecteurs, géométrie euclidienne...)

Points forts :

- Simplicité
- Evolutivité

Difficultés rencontrées :

- Trouver des exemples en DirectX

Introduction

La génération des maps se devait d'être simple, en attendant un hypothétique générateur de maps que nous n'avons d'ailleurs pas eu le temps de réaliser. Dans ce but, un parseur de fichiers textes (détaillé en 6.3) a été développé afin de permettre une génération facile de maps et leur affichage. Nous ne détaillerons pas les fonctions utilisées par DirectX mais juste le principe de cet affichage...

Principe

Les fichiers maps pouvaient contenir n'importe quelle sorte d'info, la flexibilité du parseur permettant de réellement tout faire. Par conséquent on y entre les informations relatives aux modèles 3DS et aux textures que l'on souhaite charger. Une fois ces éléments chargés, on va chercher à les afficher. On instancie donc les modèles 3DS et ils deviennent des objets de la scène en cours. On peut leur assigner une vitesse initiale, une position etc... En ce qui concerne les textures, elles viennent décorer les polygones dont on spécifie les coordonnées spatiales ainsi que les coordonnées de textures qui vont spécifier l'application de la texture au polygone.

Exemple

```
/* Indique le nombres de textures et de modèles
que l'on va charger */
index {
  0, 0;
}

texture {
  /* Charge la texture du fichier "42.jpg"
  et lui assigne l'ID 0
  0, "42.jpg";
}

meshes {
  /* Charge le modèle "tree.j3DS"
  et lui assigne l'ID 0 */
  0, "tree.3DS";
}

object {
  /* Instancie le mesh tree
  d'ID 0, de classe 1, de position (0,0,1)
  et de vitesse (0,0,0) vitesse,
  de propriétaire d'id 2*/
  0, 1, 0,0, 1, 0, 0, 1, 2;
}

triangle {
  /* dessin d'un triangle ayant pour sommets
  (0,0,0) ; (0,1,0) et (1,1,0)
  avec la texture ayant l'ID 0 */
  0,0,0, 0,1,0, 1,1,0, 0
}
```

Conclusion

Ces informations combinées à la librairie DirectX permettent l'affichage des maps. La procédure de création de map, même si un peu longue, reste simple et nous a permis d'élaborer la map de la soutenance. Ces procédés ont aussi été appliqués aux fichiers de modes de jeu.

1.2 Loader 3DS

Réalisation : Laurent

Période : Octobre - Juin

Prérequis :

- Connaissances en Direct X
- Parseur de fichier

Points forts :

- Puissance de 3DS
- Simplicité

Difficultés rencontrées :

- Trouver une référence des codes utilisés par 3DS

Introduction

Rencontrant certains problèmes avec les *mesh* (modèles) DirectX et devant l'impossibilité d'en gérer autrement qu'en utilisant 3DSMax, je me suis attelé à la production d'un loader 3DS. La tâche ne s'est pas rêvélée être évidente par l'absence de sources Delphi / DirectX. Les informations du fichier "3DSinfo.txt", disponible sur le web m'ont permis de mener à bien cette opération, et c'est vraiment cool quand ça marche !

Parsage des fichiers 3DS

La première question que l'on peut se demander étant "mais qu'est-ce qu'un fichier 3DS ?" je vais tout d'abord présenter le mode de fonctionnement de ces fichiers. Les fichiers 3DS présentent une structure arborescente, ce sont même des arbres planaires généraux. Utiliser une fonction récursive de traitement pourrait donc être judicieux, mais en réalité il n'est rien. Puisque le parcours du fichier reste linéaire une simple procédure itérative va se charger du travail.

Le fichier 3DS est composé de "chunks", chaque chunk est un arbre, éventuellement vide ou même de racine non étiquetée. Dans le fichier 3DS, ces chunks commencent tous sans exception par 6 octets : un word pour l'id et un double word pour la taille du chunk. L'id nous renseigne sur le type de données que cet arbre comprend, et la taille nous permet de connaître exactement le nombre d'octets à récupérer, voire à sauter si on les juge inutiles.

Cette lecture de l'id et de la taille est la première étape de la boucle principale de la fonction itérative. On va ensuite utiliser le contenu du fichier selon cet id. Par exemple **$4D4D** est l'id de l'arbre général du fichier tout entier. Sa taille est définie récursivement par **0 + taille(sous-arbres($4D4D))**.

Chaque chunk peut avoir une structure interne comprenant des valeurs. Il suffit donc simplement de lire les valeurs de ces chunks pour récupérer les informations du fichier 3DS, c'est aussi simple que cela. La plus grande difficulté fut de trouver une liste de ces chunks comprenant aussi une référence de leur structurer, autrement c'est un peu compliqué à parser !

Utilisation du contenu parsé

Parser un fichier 3DS c'est bien, l'utiliser c'est encore mieux ! Le point positif c'est que DirectX et 3DSMax utilisent sensiblement les mêmes notions, avec le même nom. La gestion des materials (propriété définissant la réaction d'une surface soumise à une lumière), des couleurs ou des formes géométriques se fait aisément une fois que l'on maîtrise DirectX. On remplit différents tableaux de coordonnées, on charge diverses textures et matériaux, et on demande à DirectX de nous générer un mesh avec tout ça.

Conclusion

Le résultat est vraiment plaisant, les modèles sont beaux, réagissent à la lumière et surtout sont disponibles sur le net, malheureusement la majeure partie d'entre eux sont payants...

Le loader de fichiers 3DS n'est pas compliqué en soi une fois que l'on trouve une référence de la structure de ces fichiers et que l'on maîtrise le moteur graphique de notre application.

2 Physique

2.1 Octree

Réalisation : Florent

Période : Octobre - Décembre

Prérequis :

- Listes doublement chainées
- Algorithmes de parcours d'arbre
- Bases de POO

Points forts :

- Vitesse d'accès (parcours itératif en coordonnées relatives)

Difficultés rencontrées :

- Destruction propre de l'arbre et de son contenu
- Adaptation d'un arbre général à l'application

Introduction

Un jeu de type FPS doit être capable de gérer une multitude d'objets physiques et leurs intéractions. Bien qu'un monde dans lequel les balles et autres projectiles ne blessent personne serait fantastique, il est difficile d'imaginer un jeu vidéo négligeant délibérément cet aspect. Ainsi, il nous a fallu nous lancer dans l'énorme tâche qu'est la gestion des collisions. En terme simple, gérer les collisions dans un projet tel que le notre revient à :

- Prendre tous les objets un par un.
- Chercher tous les objets qui lui sont proches.
- Regarder si par hasard il y a collision avec l'un d'entre eux.
- Agir en conséquence (destruction, arrêt, rebond,...).

Seulement, la notion de proximité est tous simplement inexistante en informatique. Il faut l'implémenter d'une manière ou d'une autre, en gardant à l'esprit que le nombre de calculs, et donc la performance globale de l'application, décroît rapidement. Nous avons donc cherché une structure capable de donner, pour tout objet de notre espace de jeu, ce qui se trouve dans une certaine zone autour de cet objet, et qui donc est susceptible d'intéragir avec lui. On peut distinguer trois structures susceptibles de répondre à notre attente : le tableau, dont l'inconvénient majeur est qu'il nécessite un espace mémoire contigu important ; le tableau chainé, qui perd les avantages de l'accès direct ; et l'arbre, pour lequel les temps d'accès aux feuilles est fixe, mais qui demande un espace mémoire plus important. Pour pallier les problèmes d'espace mémoire, nous avons donc opté pour le bonzaï.

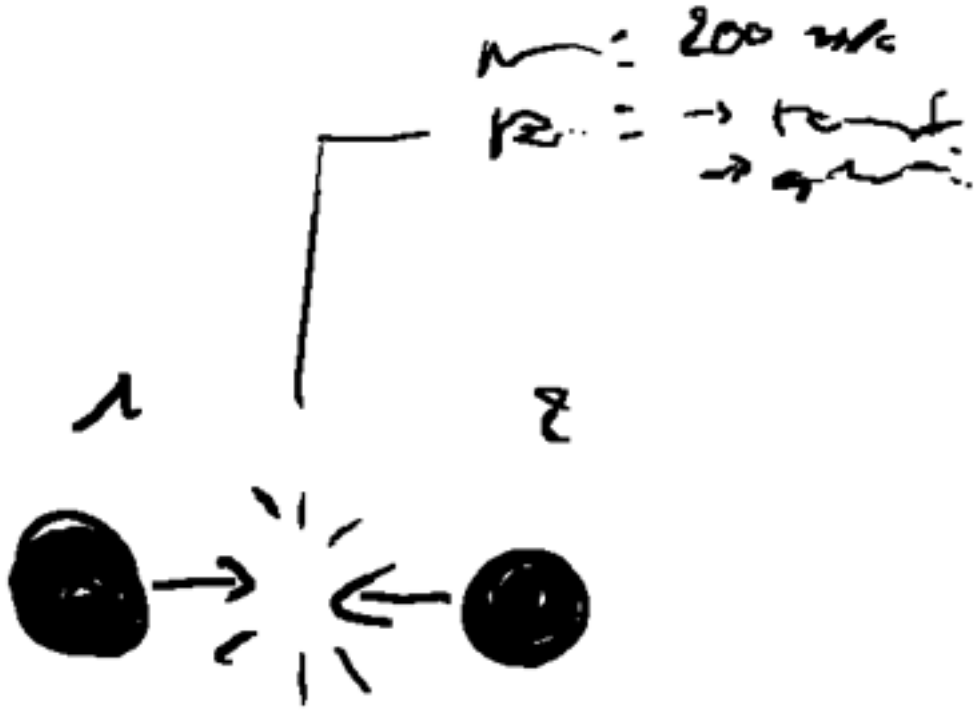
Le principe de l'octree

Il ferait mauvais genre de rappeler ici le principe d'un arbre, donc venons-en directement aux spécificité de l'octree. Contrairement aux arbres binaires, chaque sous-arbre ne possède non plus deux, mais huit fils.

Dans une représentation en 3D, il est facile de visualiser l'architecture d'un tel arbre : La zone de jeu, un simple cube, sera divisé en huit cubes de taille identiques qui formeront ses huit fils. Chacun d'eux sera ensuite divisé en huit, et ainsi de suite, jusqu'au niveau de précision désiré. Nous travaillerons ici sur un octree complet, dont toutes les feuilles sont au même niveau. Ainsi, la zone de jeux est divisée en une multitude de petits registres qui sont les feuilles de notre octree. Dans ces registre sont stockés des pointeurs vers tous les objets dont ils contiennent le centre. Ainsi, récupérer les voisins d'un objet donné revient à lister tous les objets présents dans le même registre que celui-ci, et les registres adjacents si besoin.

L'implémentation

Pour des raisons pratique, notre implémentation a été adaptée à des besoins spécifiques, notamment la recherche d'une optimisation de temps de parcours. Ainsi, chaque feuille de l'octree connait la position de toutes les feuilles adjactente. De plus, l'orientation objet étant particulièrement gourmande en ressources, que ce soit vitesse ou temps de calcul, elle n'a été conservée que pour la structure globale : les millions potentiels de sous-arbres, ansi que les registres situés dans les feuilles (listes chainées), ont été recodés de manière indépendante de l'implémentation objet. Pour des structures de ce type, et présentes en si grand nombre, les procédés de vérification inhérentes à l'objet représentent une perte de performance de plus de 30%, pour un résultat absolument identique. Des procédures de création et de destruction ont bien évidemment été écrites afin de ne permettre aucune fuite de mémoire.



2.2 Collisions

Réalisation : Florent

Période : Janvier - Mars

Prérequis :

- Mécanique newtonienne
- Equations mathématiques (prévision du déplacement des objets)
- Utilisation de l'octree
- Gestion d'effets (voir Gameplay - Effets)

Points forts :

- Test rapide et général pour tout objet ou polygone
- Test prévisionnel : aucune collision manquée quelque soit le 'pas' des objets physiques

Limitations :

- Collisions uniquement basées sur des sphères

Difficultés rencontrées :

- Prévoir la portion exacte de l'octree à parcourir en fonction de la vitesse des objets

---

Introduction

Le moteur de gestion des collisions a été en développement depuis la première soutenance en Janvier. Ce que nous attendons du moteur n'est pas une gestion de collisions entre modèles très précise, puisque pour rendre le jeu équitable, la zone de dommage des personnages dans les FPS est identique pour tous les joueurs. Par contre, il nous faut pouvoir traiter un grand nombre d'objets simultanément sur scène, afin de rendre le jeu plus immersif.

Aini, le module physique qui traite les collisions fonctionne de manière générale sur tous les objets de l'environnement, et ce, en trois temps :

Traitement des messages

On interprète d'abord les messages provenant du réseau, ce qui permet de savoir si un joueur se déplace, tire, ou quitte le jeu. La gestion du temps et du nombre de rafraichissements étant gérée par le module réseau, donc de manière centralisée par le serveur de jeu, le rafraichissement de l'environnement de jeu est déclenché par la réception du message correspondant.

Calcul des collisions

Le module parcours alors une liste d'objets contenus dans une instance d'environnement global (voir Fig.1) :

- Octree : Pointeur vers l'octree qui contient tous les objets instanciés, ainsi que les polygones simples composant la map, triés selon leur emplacement dans l'aire de jeu.
- PhyNorm : Liste chainée de tous les objets soumis aux collisions (par opposition aux objets aux trajectoires précalculées, comme les particules).
- CIDTable : matrice à deux dimension contenant des pointeurs de procédure, qui déterminent selon la classe de deux objets qui entrent en collision, leur comportement (rebond, destruction, perte de vie,...).
- CIDPTable : vecteur contenant des pointeurs de procédures, utiles cette fois lorsqu'un objet rencontre un polygone de la map.

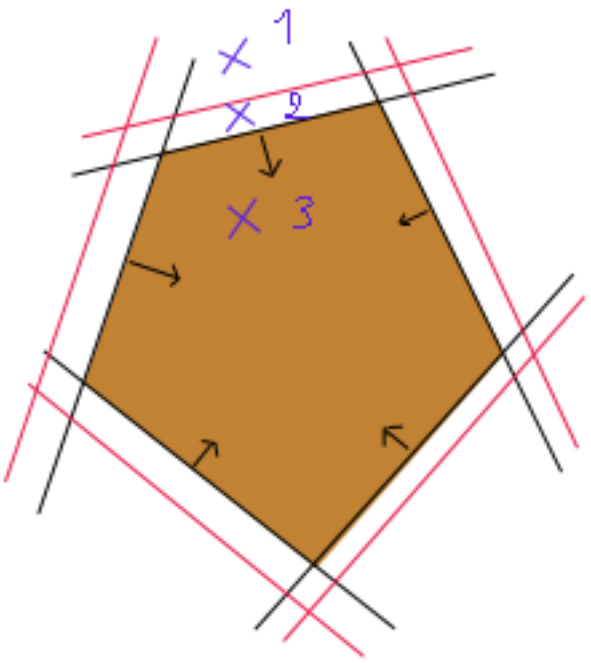
C'est donc la liste PhyNorm qui est parcourue, et pour chacun des objets qu'elle contient on procède ainsi :

```
TEnvi = class
private

public
  IDLoc:       Cardinal;
  Gniii:       pTGniii;
  Octree:      pTOctree;
  Menu:        pTMenu;
  HUD:         pTHud;
  Polys:       pTDChaine;
  PhyNorm:     TDChaine;
  PhyAff:      TDChaine;
  PhyForces:   TDChaine;
  Camera:      TCamera;
  CIDTable:    TCIDTable;
  CIDPTable:   TCIDPTable;
```

Fig. 1 – (seule une partie de la classe est détaillée ici)

- Récupération de la liste des objets situés dans les registre proches de l'octree (seul le centre de l'objet étant répertorié dans l'octree, le nombre de registres parcourus dépend du rayon dudit objet, et du rayon de l'objet le plus grand présent).
- Détection de collision prochaine entre la sphère représentant l'encombrement de l'objet traité et chacunes des sphères représentant les objets proches. Il s'agit de résoudre l'équation déterminant le temps auquel deux se trouveront à une distance inférieure à la somme de leurs rayons. Si ce temps est inférieure à 1 (rafraichissement de position), la collision est imminente et est traitée immédiatement, en appelant la fonction de résolution située dans la CIDTable.
- Récupération de la liste des polygones proches (qui cette fois, sont insérés dans tous les registres qu'ils coupent)
- Détection de collision entre l'objet et chacun des polygones. Il faut ici non seulement déterminer à quel moment l'objet entrera en contact avec le plan de polygone, mais également si il y aura contact entre l'objet et le polygone, c'est à dire si le point de contact avec le plan se trouve à une distance inférieure à un rayon de l'intérieur du polygone :



A l'aide de produits scalaires et des normales aux droites des cotés situées dans le plan du polygone, on détermine pour chacun des cotés si le point est suffisamment proche ou non de l'intérieur (entre les droites rouges). On a ainsi trois registres correspondants à :

1. Pas de collision.
2. Collision de coté.
3. Collision frontale.

La collision est ensuite traitée en fonction de l'ID de la classe de l'objet.

– Si un des objets doit etre détruit, il est ajouté à une liste de destruction, afin que sa suppression de perturbe pas la détection d'autres collisions. En dernier lieu, la liste est parcourue et les objets supprimés.

Mise à jour des positions

Les caractéristiques mécaniques des objets sont finalement mises à jour, c'est à dire position, vitesse, accélération, en fonction de la classe de ceux-cis.

2.3 Forces

Réalisation : Florent

Période : Février

Prérequis :

- Mécanique newtonienne
- Utilisation de l'octree
- Gestion d'effets

Points forts :

- Forces applicable à une zone variable (proximité d'un point)
- Effets variables

Description

La grande souplesse (moui, toute relative... NDC[3]) de l'octree et du moteur physique a permi d'implémenter divers autres trucs cools qu'on voit dans la vrai vie. Genre, ici, des forces d'attraction ou de répulsion qui peuvent s'appliquer sur tout ou partie de l'environnement physique. Tiens, par exemple, la gravité. Topo.

Gravité

"Laisse le par terre, 'tombera pas plus bas..."

Comme chacun sait, ou devrait savoir, la gravité terrestre, ca se traduit par une accélération de 9.8 m/s environ à la surface de la terre. Pour faire original, on a donc implémenté une force verticale de 9.8 truc/machin, qui s'applique sur à peu près tous les objets mobiles (sauf ceux qu'on veux magiquement maintenir flottant dans l'espace... mouack). Résultat, on est pris de maux de coeur, le monde s'écroule autour de nous, et bref, beaucoup plus dur de viser maintenant que les projectiles de 30kg lancés à la mains ont la facheuse tendance de vouloir redescendre.

Mais l'on a fait bien plus fort encore...

L'électro-gravito-magnétisation

Peut-être que quelqu'un s'en souvient, j'en ai parlé un peu plus haut, un super octree balèze a été codé. Autant s'en servir, non ? Bref, ici, il s'agit

[3]Note du Codeur

de mettre en place des forces qui ne s'appliquent que sur partie réduite de l'environnement, et qui attirent vers, ou repoussent de n'importe quel point de l'espace. Comme le reste, ça se définit dans les fichiers de map, ça se charge tout seul, et ça nous fait perdre trois quart d'heure à s'amuser à mettre en orbite les différents projectiles (avec succès... joie !)

Deux trois détails techniques

De même que pour les détections de collision, avec la force concernée, on fait un parcours de l'octree relativement au point d'application et on récupère tous les objets suffisamment proches pour être soumis à la force en question. Puis on applique un effet variable à chacun de ces objets, bien souvent une modification de leur vitesse (représentée par un vecteur à trois dimensions)

Là ou ça coince

Manque de chance, on a pas vraiment eu de problème avec cette partie du projet... Codée vite et bien, le rêve, quoi.

3 Le son

Réalisation : Laurent & Renaud

Période : Février - Mars

Prérequis :

- Prise en main de FMod

Points forts :

- Simplicité de charger et de jouer un son n'importe où dans l'application

Difficultés rencontrées :

- Prise en main de FMod

Introduction

Nous voulions utiliser DirectSound au départ, mais aux vues de l'absence quasi totale d'exemples de sources en Delphi, notre choix s'est réorienté vers la bibliothèque FMod qui est très simple d'utilisation et qui contient tout une flopée de fonctions très pratiques.

Avec FMod il est assez Simple de charger un son, par exemple si l'on veut charger le son "*mon_son.wav*" on procéde ainsi :

```
FSOUND_SAMPLE_LOAD(channel,'mon_son.wav',Mode de Lecture,0,0) ;
```

Pour un mp3 c'est un peu différent car il s'agit d'un stream et le charger en mémoire demanderait trop de place. On utilise alors :

```
FSOUND_STREAM_OPEN('mon_mp3.mp3',Mode De Lecture,0,0) ;
```

Une fois chargés c'est tout aussi simple de les jouer :

```
FSOUND_PLAYSOUND(Channel, Mon Sample) ;
FSOUND_STREAM_PLAY(Channel, Mon Stream) ;
```

Pour détruire les son :

```
FSOUND_SAMPLE_FREE(Mon Sample) ;
FSOUND_STREAM_CLOSE(Mon Stream) ;
```

Il est bien sûr possible de jouer simultanément le même son sur plusieurs channels différents.

Principe

Tous les samples dont nous avons besoin sont au lancement du jeu ou de la partie chargés dans une liste. Si l'on veut jouer un son il suffit d'accéder à l'emplacement du son dans la liste et de le jouer.

Comme toutes les parties du projet, les évènements sonores sont contrôlées par un Thread. Le Thread contient une boucle qui actualise les informations dans la liste de sons en train d'être joués en fonction des informations transmises par le moteur physique ainsi que la position du joueur.

Quand on joue un son, un channel lui est attribué. Pour pouvoir gérer le son - le mettre en pause, actualiser sa position dans l'espace - il est nécessaire de créer une nouvelle liste dans laquelle figurent les sons en train d'être joués. Dans le cas des stream, ceci ne libérant pas le channel une fois terminés ; il est nécessaire de tester si sa lecture est terminée et dans ce cas, on libère le channel. Quand nous n'avons plus besoin des sons, c'est à dire à la destruction du thread, ou quand un MP3 est fini, une procédure les détruit.

Son 3D

FMod permet de gérer facilement un environnement sonore en 3D. La physique fournit en permanence au thread la position du joueur et, pour chaque son, leur position dans l'espace. Ainsi le volume et les balances sont directement contrôlées par FMod lui même.

Ainsi on peut positionner notre joueur dans l'espace grâce à la procédure

```
FSOUND_3D_LISTENER_SETATTRIBUTES(position, vitesse,
      orientation) ;
```

Pour le son il s'agit de :

```
FSOUND_3D_SETATTRIBUTES(Channel du son, position,
      vitesse) ;
```

On peut aussi déterminé jusqu'à quelle distance le son est à pleine puissance et jusqu'à quelle distance il sera audible grâce à cette dernière fonction :

```
FSOUND_3D_SETMINMAXDISTANCES(Channel, DistanceMin, DistanceMax) ;
```

Conclusion

Le résultat est assez convaincant avec FMod, puisqu'en un minimum de fonctions il nous a permis de réaliser entièrement, ou presque, ce que nous souhaitions obtenir.

4 Gameplay

4.1 HUD

Réalisation : Laurent

Période : Mars

Prérequis :

- Prise en main de Direct3D

Points forts :

- Aucuns

Difficultés rencontrées :

- Problème de compatibilité avec différentes cartes graphiques

---

Introduction

Le "HUD" est l'interface entre le joueur et la partie. Il permet de connaître en temps réel de connaître sa vie, son énergie, le stock de munitions, le nombre de joueurs encore en vie... bref si l'on est dans la merde ou pas !

Les sprites

Pour afficher les différents items on aurait pu constamment les afficher devant la caméra, mais il y a bien plus intelligent : repasser en 2D.

Direct X permet un affichage très simple de la 3D, et ce, à l'aide des "sprites". Il était un temps pas si lointain ou Mario n'était pas un ensemble de triangles, mais juste une image de quelques pixels appliquée à une surface, à savoir la télé.

Le principe reste le même, mêeme si depuis depuis la technique a évolué. Nous appliquons donc nos textures sur une surface 2D, notre écran. Le système est donc adapté à ce que nous comptions faire, à savoir afficher un HUD digne de ce nom.

Les données physiques comme le niveau d'énergie ou le nombre de munitions sont stockées dans une variable connue par l'objet HUD, en effet un pointeur vers cette variable est une propriété de cet objet. Il reste ensuite à

afficher les différents éléments. On peut de plus afficher du texte, en utilisant une texture par caractère.

Les inconvénients

Bien qu'*a priori* ce système ne demande rien de particulier puisque très facile à mettre en place, il nécessite néanmoins une importante place dans la mémoire vive. En effet, chaque texture est mise en mémoire afin d'être utilisée.

De plus, la qualité des textures ne peut pas être maximale, comme en bitmap par exemple ; cela prendrait beaucoup trop de place. De fait, on utilise un format assez pratique puisque gérant la transparence : le PNG. La qualité est très bonne et la transparence peut alors être pleinement exploitée.

Compatibilité

Un problème a été rencontré avec ces sprites. Bien que leur manipulation soit basique et même nécessaire pour tout jeu 3D, il semblerait que la librairie DirectX de Clootie ne fonctionne pas sur tous les PC. Ainsi un de nos trois PC ainsi que les PC en salles machines affichent mal les sprites. Les menus semblent désorganisés, les textures floues ou mal placées. Le même code compilé en C++ ne génére lui aucun problème. Par conséquent si jamais vous constatez ce bug, sachez que le problème ne vient pas de nous ! (comme si !)

4.2 Menus

Réalisation : Florent/Renaud

Période : Juin

Prérequis :

- Sprites sous DirectX
- Structures de données arborescentes
- Lecture Ecriture dans les fichiers texte et INI.

Points forts :

- Flexibilité de la création des menus et sous-menu a partir du fichier
- Rendu visuel

Difficultés rencontrées :

- Comprehension du système déjà en place

Introduction

La structure du menu est un arbre dont la racine contient le menu principal et dont les fils sont des items qui peuvent contenir des sous menu. Il s'affiche à l'ecran avec des textures qui changent d'états durant la navigation.

Le Type TMenu

Le type contient les propriétés relatives au menus en général comme les sprites et les textures utilisées durant toute l'exécution du menu (arriere plan, logo, flèches etc ...).

Il contient les procedures appelées les items du menu.

Arborescense

Le menu racine contient un certain nombre d'items, chacun ayant un certain nombre de propriétés (liste non exhaustive) :

- La texture propre l'item
- La largeur de la texture, necessaire au positionnement des flèches
- Un pointeur vers le père de l'item
- Une liste chainée contenant les items du sous menu associé éventuel
- si l'item est disponible
- si la texture doit etre mis en surbrillance

– L'action à effectuer (Ouverture d'un sous-menu, changement de configuration, démarrer une partie ...)

Si l'on veut afficher le contenu d'un sous menu associé à un item, il suffit de définir l'item comme racine.

Affichage

Pour obtenir un bon rendu visuel, l'affichage se fait en fonction : du nombre d'items du menu, de la résolution d'écran et de la taille des textures.

A chaque raffraichissement, la liste des Items est parcourue, et les textures associées sont affichées en fonction des propriétés(surbrillance, disponibilité... ou pas).

Navigation

La navigation se fait par capture d'évènements clavier par *Direct Input* gérés dans le Thread Effecteur. De cette façon on peut se déplacer dans la liste d'items, en modifier les propriétés et déclencher les actions associées.

Le fichier menu

La structure du fichier est la suivante :

```
Item1
fichier texture Item1
Largeur texture Item1
Action Item1
Disponibilité Item1
    {
    SubItem1
    fichier texture subItem1
    Largeur texture subItem1
    Action subItem1
    Disponibilité subItem
        {
        subsubItem1
        fichier texture subsubItem1
        Largeur texture subsubItem1
        Action subsubItem1
        Disponibilité subsubItem1
```

```
            subsubItem2
            fichier texture subsubItem2
            ...
            }
        subItem2
        ...
        }
Item2
...
```

Une procedure récursive parcourt le fichier et construit le menu en fonction. Il est donc trés simple par la suite d' ajouter des items et d'en supprimer.

Conclusion

Au final nous avons un menu esthétiquement satisfaisant, dans lequel on navigue de façon intuitive.

4.3 Contrôles

Réalisation : Laurent & Florent

Période : Février - Mars

Prérequis :

- Prise en main de DirectInput

Points forts :

- Rapidité

Difficultés rencontrées :

- Difficile de trouver de l'aide pour Delphi

Introduction

Le système de messages Windows offrait un environnement facile à utiliser mais présentait de gros inconvénients : les temps de latence. Nous nous sommes donc tournés vers une alternative performante et finalement aussi simple, à savoir DirectInput.

Principe

L'effecteur est un thread rapide, prévu pour etre exécuté jusque 100 fois par seconde. Bien que cette limite soit un peu exagérée, il faut noter que le déplacement de la caméra dans un FPS est primordiale pour avoir une vision fluide de l'environnement. Ainsi nous avons fait le choix de dissocier le rafraichissement de caméra du rafraichissement par le moteur physique du joueur lui même. L'effecteur a donc la charge de mettre à jour la direction de la caméra, et de prendre note des actions du joueur sur le clavier et la souris.

On suit donc tout d'abord la progression de la souris en coordonnées relatives. Et c'est bien là l'avantage de Direct Input. Si l'on obtenait les coordonnées en absolu, il y aurait quelques soucis quand le curseur arriverait au bord de l'écran (même si on est en dual screen...) Puisque Direct Input gère le déplacement en relatif, nous n'avons pas ce problème car nous connaissons exactement la quantité de mouvement de la souris suivant les axes X et Y. Nous y avons bien entendu ajouter la gestion des boutons gauche, droit, de la molette ainsi que de boutons auxiliaires. Concernant le clavier, l'état de chaque touche (enoncée ou non) est récupéré dans un tableau d'une centaine de shorint correspondant à chacune des touches.

```
// Quit
if (Controls^.Quit.Keys[0]^ = 128) or
   (Controls^.Quit.Keys[1]^ = 128) then
begin
  Gniii^.continue := False;
end;
```

Bien que le parcours d'un tableau de cette taille ne soit que peu gourmand en ressources, le système de pointeurs permet un controle astucieux des commandes, et une personalisation des controles tout ce qu'il y a de plus aisée : Le comportement de chacune des action est défini dans le programme, et à chaque action est associée une adresse mémoire, celle contenant l'état de la touche qui lui est assignée dans les paramètres de configuration, adresse qui se trouve appartenir au tableau dans lequel est mis à jour l'état du clavier par DirectInput. On peut donc facilement modifier les contrôles, même en temps réel.

4.4 Effecteur

Réalisation : Florent

Période : Janvier - Juin

Prérequis :

- Gestion des contrôles personnalisés
- Entrée texte avec DirectInput
- Gestion des threads

Points forts :

- Exécution indépendante du reste de l'application, donc caméra fluide

Difficultés rencontrées :

- Entrée du texte de manière alternative

Introduction

Le fonctionnement du thread effecteur est très lié à la gestion des contrôles. Il gère l'interprétation des contrôles et l'envoi des différentes ordres vers le serveur de jeu (par l'intermédiaire du module client). Son fonctionnement mérite d'être quelque peu détaillé.

Fonctionnement

A la base, l'effecteur, c'est comme une grosse boucle while true do things. Ca tourne en arrière plan du début à la fin de l'application et grève les performances générales. Sauf qu'ici c'est nécessaire. On a donc une boucle, qui selon l'état du jeu (Menu ouvert, demande d'entrée texte, jeu en route), lance des boucles de gestion souris et clavier différentes. La gestion des contrôles est détaillée dans une autre section, mais un autre aspect important est l'entrée de texte pendant la partie.

Système de Chat

Comme dans tout bon jeu qui se respecte, les joueurs doivent pouvoir communiquer entre eux, histoire de mettre en place des stratégies minutieuses, de coordonner des attaques chirurgicales, ou bien plus souvent, de raconter des débilités caractéristiques des nouvelles générations (Ou pas.). Bref, ici, c'est simple. On a une file (circulaire !) de messages, dont on affiche qu'un certain nombre d'éléments, pendant quelques secondes. Là ou ça se

complique, c'est quand il faut écrire ces messages. En effet, l'utilisation de DirectInput rend moins aisée la gestion des messages clavier windows (que l'on a dans un premier temps voulu éviter par soucis de rapidité). Il faut donc interpréter les appuis de touches récupérés avec directInput, simuler un délai de répétition, et traiter les caractères récupérés. Quelques API windows font une partie du travail, mais curieusement, c'est pas évident d'avoir un truc fonctionnel. Bref, une partie du système a été réécrite façon DGP.

En Gros

L'effecteur, il sert ; l'effecteur, il marche ; l'effecteur, on oserait dire qu'il poutre, mais on se retiendra, encore trop de frustration par là.

4.5 Gestion d'effets

Réalisation : Florent

Période : Février - Avril

Prérequis :

- Pointeurs de procédures (abstraites)
- Allocation/libération mémoire avancée (taille de données variable)
- Une patience énorme

Points forts :

- Permet de redéfinir facilement des styles de jeu différents
- Codage souple : chargement dans un fichier .gmd

Difficultés rencontrées :

- Ajout/suppression à chaque instance d'objet
- Ack... Déréférencer un pointeur... euh non, chaque pointeur...

Introduction

On en a parlé un peu partout, un des points forts de ce projet est que l'on a tenté de faire quelque chose qui pourrait être repris par la suite, ou du moins, un modèle d'architecture assez souple qui pourrait servir de base à n'importe quel type de jeu. Bref, on a un système de classe qui permet de définir différentes actions pour chaque évènement dans l'environnement ("Wow, matte les deux sphères là bas... *POUM* ouaiiis !!"). Une banque d'effets a donc été créée pour permettre d'assigner des comportements divers, variés, et pourtant si prévisibles.

'M'en ca marcheuh ?

Simple... du moins à dire. Chaque effet est en fait un enregistrement à trois champs. Le premier, un entier, représente une taille de données... boarf, pas important. Le deuxième, un pointeur. Ouais, genre, une adresse mémoire quoi. Bof. Le troisième, une autre pointeur, de procédure cette fois. Et c'est tout. Blah. Mais comme on est ingénieux, ca marche :
Lorsqu'on veux créer un effet, on appelle une fonction de création spécifique qui écrit dans un p'tit coin de la mémoire tout les paramètres nécessaires à l'exécution de l'effet concerné. Faut savoir que c'est souvent plus ou moins abstrait, du genre "endommage l'autre de 3HP". On comprend mieux quand on applique ça à une collision entre deux objets. Bon, pour trouver quels

sont les objets concernés, on se sert de méthodes pas très catholiques, voire hérétiques (qui a dit barbares ?), bref, des pointeurs vers pointeur vers le rapport de collision, qui possède lui les adresses des différentes entités impliqués, la date et heure à laquelle la collision va se produire (? !), et autres joyeuseries. Extra details in the source code. Good luck mate.

J'résiste pas...

```
  TGenEffect     = procedure (_arg: pointer);

  TEffect = record
    arg: pointer;
    siz: word;
    fnc: TGenEffect;
  end;

Type
  pTObjValClcData = ^TObjValClcData;

  TObjValCLCData = record
    obj: pointer; // pointeur vers pTOPhy
    val: integer;
    clc: pTDChaine;
  end;

// endommage l'objet concerné
procedure ObjDamage(_arg: pointer);
begin
  with pTObjValClcData(_arg)^ do
  begin
    pTOPhy(obj^)^.life := pTOPhy(obj^)^.life - val;
    if pTOPhy(obj^)^.life <= 0 then
    begin
      clc^.FAddItem(pTOPhy(obj^));
      pTOPhy(obj^)^.idq := True;
    end;
  end;
end;


function  ObjDamageCreator(Gniii: pTGniii;
```

```
                _cd: pTCollData; arg:string): pTEffect;
var
  o: pTObjValClcData;
begin
  new(o);
  if LowerCase(getString(arg)) = 'other' then
    o^.obj := @(_cd^._o2)
  else
    o^.obj := @(_cd^._o1);
  o^.val := getInteger(arg);
  o^.clc := _cd^.clc;
  new(Result);
  Result^.arg := o;
  Result^.siz := SizeOf(TObjValClcData);
  Result^.fnc := ObjDamage;
end;
```

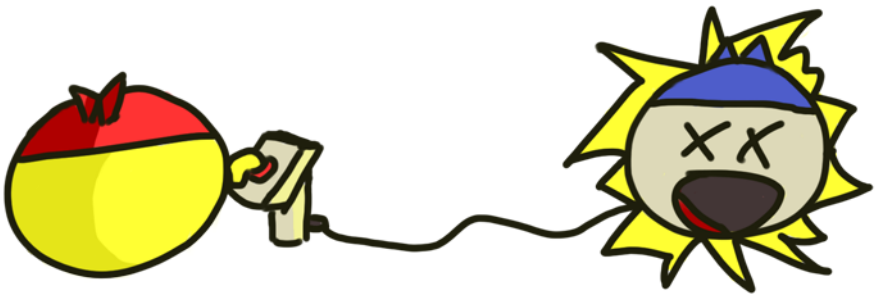
Qui permet, par exemple lors d'une collision, d'endommager un des objets concerné. CQFC EER[4]

Résultat

Après beaucoup de temps passé à traquer les erreurs de syntaxe, à se forcer à coder des fonctions de duplication et de destruction qui fonctionnent, et un peu de ras-le-bol, ça a fini par marcher. Tout ça pour ça... mais que d'expérience gagnée pour nos travaux futurs... 'Fin j'espère...

[4] *Ce qu'il fallait coder efficacement et rapidement*

5 Reseau



Réalisation : Renaud/Florent

Période : Avril-Juin

Prérequis :

- Fonctionnement et architecture d'un reseau
- Programmation orientée objet

Points forts :

- Aucuns

Difficultés rencontrées :

- Récupération des évènements en console

Introduction

Pour réaliser le reseau nous avons utilisé les classes *TClientSocket* et *TServerSocket* de Delphi. Ces classes comprennent des propriétés et des procédures permettant de gérer facilement les connections de sockets sous windows.

Fonctionnement

La classe Client comprend un objet TClientSocket qui se connecte au serveur et permet d'envoyer des paquets ainsi qu'un objet TServerSocket qui permet de receptionner les paquets provenant du serveur.

La Classe Serveur comprend un objet TServerSocket qui écoute les éventuelles connections de clients et une liste d'objets Clients. Lorsque qu'un client se connecte, on crée un objet client dans la liste qui va ouvrir une connection avec l'objet TServerSocket du client fraichement connecté.

Récupération des évènements

Les Classes TClientSocket et TServerSocket gèrent des évènements qui déclenchent des procédures. Par exemple quand un client se connecte ou se deconnecte, quand une connection est établie, quand des paquets doivent etres réceptionnés. //

Ces classes sont à l'origine faites pour etre utilisées avec des *forms* qui récupèrent automatiquement les évènements et déclenchent les procédures associées. Comme nous n'utilisons aucune *form* dans notre projet il a fallu une astuce qui n'a pas été facile a trouver :

A la création de notre objet, il faut associer manuellement les évèvements des sockets à des procédures de sa classe, comme dans les exemples suivants :

```
Client.OnConnect    := ProcedureConnection
Client.OnDisconnect := ProcedureDeconnection
Serveur.OnClientConnect := ProcedureConnectionDeClient
...
```

Maintenant, il s'agit de permettre le déclenchement de ces évènements, mais ce qui se fait automatiquement avec l'utilisation de forms, a dû être implémenté manuellement : Ce sont des threads créées pour l'occasion qui sont chargés de vérifier périodiquement si certains de ces évènements ont eu lieu, et de d'appeler ces procédures.

Au final, ça marche. ouf ! . Mais nous n'étions pas au bout de nos peines...

Récupération des paquets

Les composants windows gèrent seuls les connections et déconnections, ainsi que l'acheminement à bon port et dans le bon ordre des paquets transférés, mais il nous reste à encoder, relayer, et décoder toutes les actions possibles dans le jeu. Ainsi, il a fallu créer des fonctions chargées de copier dans un espace mémoire contiguë des informations éparses, et de renvoyer leur emplacement, pour ensuite envoyer de client à serveur une plage mémoire spécifique. Dans une partie multijoueur, les paquets sont centralisés sur le serveur, leur légitimité vérifiée, puis redistribués à chaque client. Une fois parvenus à destinations, ils sont de nouveau filtrés puis interprétés pour effectuer des actions dans différentes modules du jeu (chat et physique par exemple)

Conclusion

L'implémentation du réseau dans un projet ne se résume pas à la bête recopie d'exemples. Elle doit être prévue à l'avance, et pensée de manière approfondie. En ce qui nous concerne, le réseau fonctionne, mais de manière pas toujours optimisée et un peu brouillonne.

6 Divers

6.1 Système de classes

Réalisation : Florent

Période : Février - Juin

Prérequis :

- POO
- Parseur de fichier
- Effets

Points forts :

- Système adaptable

Introduction

Entre la seconde soutenance et la suivante (troisième, donc...), beaucoup de travail a été effectué niveau gameplay, c'est à dire qu'une fois les procédés techniques mis en place, le but du jeu est alors de le faire (le jeu, donc). 'Fin bref, tâche pas si facile nécessitant bien souvent des modifications conséquentes de la structure interne du projet. Si peu facile, d'ailleurs, qu'on a voulu faire un système adaptable, quelque chose qui permettrait de ne pas repasser une douzaine d'heure à chasser les erreurs de typo. Et donc, On scripte !

Notions de Mod

Bien bien bien, on va faire simple : Nous avons déjà présenté lors de la seconde soutenance un système de chargement de map, qui permettait de charger polygones, textures, et d'instancier des objets dans l'environnement. Aujourd'hui, on fait encore plus fort. Un fichier *.gmd [5] contient une mine d'informations permettant faire tourner le jeu de manière totalement personnalisable :

Structure d'un fichier de mod

- **Index :** On y décrit ce qui suit, nombre de modèles, textures, sons à charger, nombre de classes déclarées, d'armes, ainsi que d'autres paramètres spécifiques

[5] *Gros Mangeur de Dattes*

- **Texture :** On liste simplement les textures utilisées en leur associant un index
- **Mesh :** De même pour les modèles
- **Sound :** ... et pour les samples audio
- **Class :** architecture de base d'un type d'objet, reprise dans les déclarations d'arme et de classe de joueur
- **Weapon :** les différentes 'armes' (moui, bon, les machins qui lancent d'autres trucs, bref pas d'quoi torturer un canard, ca reste une simple démo technique)
- **Player :** LA classe joueur. Oui, parce qu'on peut avoir plusieurs classes de joueur, avec des comportements qui peuvent être complètement différents (celui qui tient le fusil, et celui.. qui code *tsk*)
- **Forces :** Les différentes forces qui s'appliquent dans l'environnement, groovy...
- **Timer :** Pour une gestions d'évènements périodiques, mettre un peu de vie dans vos parties quoi.

Utilisation

On va encore le répéter, tout ça, c'est dans l'optique d'essayer de terminer l'année avec autre chose que de l'expérience, du code physique qui puisse être repris tel quel, par exemple... Après réflexion, l'expérience, c'est déjà beaucoup, hein...

6.2 Threads

Réalisation : Florent

Période : Novembre - Juin

Prérequis :

- POO (ouais, programmation orientée objet, on va le préciser ici...)

Difficultés rencontrées :

- Syncronisation des différents threads
- Lecture/écriture de manière sécurisée sur les même données

Intro !

On l'a suffisamment redit et répété, l'architecture de notre project se divise en plusieurs modules indépendants. Grace à l'intermédiaire des threads, nous pouvons simuler un calcul en parallèle de chacun des modules :

Comment fonctionne un thread

Un thread est un objet possédant une méthode principale Execute. Le système d'exploitation alloue tour à tour la puissance du processeur à chacun des threads lancés pendant un temps défini, ce qui leur permet d'effectuer un certain nombre d'opérations atomiques (telles que lire un emplacement mémoire, écrire sur un emplacement, effectuer un calcul ou un test). Cependant, lorsque cette période est écoulée, l'exécution du thread est stoppée, et d'autre threads peuvent utiliser la même plage mémoire. D'où un risque d'erreur élevé, si l'on ne prend aucune précaution lors de la programmation des ces objets.

Voici une illustration :

Deux threads A et B possèdent chacun un pointeur _p vers le même emplacement mémoire, ou est stocké un entier.

On ne voit apparemment ici qu'une seule opération, qui consiste pour le thread A, à ajouter 1 au contenu de l'emplacement _p, et pour le thread B, à multiplier cette valeur par 2. Pourtant, il y a en fait trois opérations dans chaque procédure :

1. la lecture de la valeur
2. le calcul du résultat
3. l'écriture de la nouvelle valeur

```
procedure ThreadA.Execute;
begin
  _p^ := _p^ + 1;
end;

procedure ThreadB.Execute;
begin
  _p^ := _p^ * 2;
end;
```

Ce qui peut engendrer le cas suivant :

```
 - Execution du thread A -
Thread A : Lecture du contenu de _p : 10;
 - Execution passe au thread B -
Thread B : Lecture du contenu de _p : 10;
Thread B : Calcul de la nouvelle valeur : 20;
Thread B : Ecriture de la nouvelle valeur : _p^ := 20;
 - Execution passe au thread A -
Thread A : Calcul de la nouvelle valeur : 11;
Thread A : Ecriture de la nouvelle valeur : _p^:= 11;

Résultat attendu : _p^ = 21;
Résultat optenu :  _p^ = 11;
```

Ansi, il a fallu redoubler d'attention au niveau du partage de la mémoire, notamment lors de l'échange de messages : un message doit être rédigé totalement avant d'être ajouté à la pile du thread récepteur (opération atomique : assignation d'un pointeur)

Comment les utilise-t-on en pratique dans le projet

En plus du processus principal, qui gère la création, la destruction de l'application et une boucle principal de traitement de messages windows, nous nous servont des threads suivants :

- Thread physique : Moteur physique, en gros, les collisions, les tirs, les forces...
- Thread serveur : Serveur basé sur l'utilisation des socket windows. Centralise les messages provenant de chacun des joureurs et les redistribue

- Thread client : Chacun des joueur en fait tourner un. Permet de faire la liaison entre le module physique et le serveur central.
- Thread d'affichage : Boucle d'affichage qui parcours l'environnement et affiche les objets qu'il contient. Comme tout est géré de manière indépendante, la fluidité d'affichage ne dépend pas des retard occasionnés par exemple par la connection à un serveur distant.
- Thread effecteur : Boucle de contrôles, qui gère les ordres de déplacement, de tir, et l'appel au menu. Quelques détails plus loin.

Difficultés rencontrées

Faire des threads qui tournent en parallèle c'est simple, les faire agir sur les mêmes données sans erreurs de synchroisation, c'est pas si évident. Finalement, on a su développer quelques astuces perso de verrou de données ou d'écriture différentielle (whoa !). Bref, plus de comportements bizarres imprévisibles.

6.3 Parseur de fichiers

Réalisation : Laurent

Période : Octobre

Prérequis :

- Aucuns

Points forts :

- Simplicité d'utilisation
- Flexibilité

Difficultés rencontrées :

- Aucunes

Introduction

Le parseur de fichier est directement lié au chargement de cartes pour le jeu mais aussi de mods. Cet outil plutôt puissant de par sa flexibilité permet d'interprêter facilement des fichiers textes.

Pour cela, nous avons choisi de développer un langage scripté, à la syntaxe C-ienne ({ , ; " /* */ // etc...) que l'on appellera le DGL, *Da Gniii Language* - ou *Le Langage qui Grince*.

Présentation du DGL

Le DGL est un langage de script. Son interprétation permettra le chargement des objets et leur placement qui seront utilisés par le moteur graphique pour l'affichage de la carte, et par le moteur physique pour les relations entre les objets et l'environnement.

Sa syntaxe est simple. On utilise des *tokens* faisant office de constructeurs. Ce sont ces derniers qui indiquent au parseur quelle opération va être effectuée. On peut les assimiler à des identifiants de fonctions.

Ces constructeurs sont séparés par des blocs d'accolades. Dans ces accolades, on y trouve des instructions, chacune se terminant par ";". Elles viennent compléter le constructeur qui accompagne le bloc, ce sont en quelque sorte les arguments qui sont progressivement passés au constructeur.

Ce langage supporte en outre deux types de commentaires : celui du langage C (/* */) et le commentaire de fin de ligne apporté par le C++ (//).

Un petit exemple rudimentaire de script

```
/* addition de deux nombres */
addition {
         30.5, 11.5; // additionne 30.5 et 11.5
         4, 5; // additionne 4 et 5
}

/* concatenation de deux strings */
concatenation {
         "Da Gniii", " Project"; // donne "Da Gniii Project"
}
```

Les constructeurs seraient ici "addition" et "concatenation" qui s'appliqueraient aux couples 30.5,11.5 et 4,5 pour l'addition et aux strings "Da Gniii", " Project" pour la concaténation. Cet exemple est plutôt simple, dans la réalité le parseur participe au chargement de textures, de maps, de modèles 3DS, de mods de jeu...

Principe

Pour parser le fichier, il faut d'abord le récupérer dans une variable. L'erreur à ne pas commettre serait de stocker l'intégralité du fichier dans cette même variable, pour des raisons de mémoire.

Il ne sert a rien de récupérer les espaces (sauf pour une exception : la string), les retours à la ligne et le contenu des commentaires.

Le fichier est lu caractère par caractère. Puisque les commentaires commencent, et même se ferment en utilisant deux caractères on va utiliser une file de deux caractères *ch1* et *ch2*. Par exemple, si *ch1* contient "/" et que *ch2* contient "*" alors on active le commentaire de ce type et on ne bufferise plus les caractères lus jusqu'à ce que *ch1* contienne "*" et *ch2* "/".

Puisque le langage fonctionne avec des constructeurs et des instructions, on peut travailler avec deux buffers qui les contiennent. On va tout d'abord récupérer le constructeur dans *current* jusqu'à rencontrer un *token* "{" La suite est en effet un ensemble d'instructions. C'est ensuite au tour du bloc d'être traité. Pour cela on va récuperer son contenu jusqu'à un *token* ";". Et cela jusqu'à ce qu'il n'y ait plus d'instructions, c'est à dire dès que *ch1* contient le token "}". Et on recommence jusqu'à arriver à la fin du fichier.

Dès que l'on récupére une instruction, on peut appeler une fonction qui va traiter cette instruction selon le constructeur courrant. Cette fonction, selon le constructeur fournit, appelle une autre fonction qui va parser l'instruction.

Pour cela, il suffit de connaître l'ordre des arguments pour les typer, et de les séparer puisque chaque argument est espacé du *token* ",".

Conclusion

Même si ce parseur a été développé tôt dans l'année (octobre), sa flexibilité et sa simplicité ne nous ont pas incités à en réaliser une version plus puissante avec les connaissances acquises tout au long de l'année.

7 Site web

Réalisation : Laurent

Période : Janvier - Mai

Prérequis :

- Bonnes connaissances en programmation web (HTML, CSS, PHP, SQL...)

Points forts :

- Site érgonomique et concis
- S'affiche bien avec tous les navigateurs
- Menu dynamique en flash

Difficultés rencontrées :

- Aucunes

Introduction

Comme annoncé dans le cahier des charges, le site web est un vecteur - non non pas ceux qu'on trie dans tous les sens - important d'informations. Nous souhaitions donc un site clair, concis et ergonomique. On y retrouve ici toute l'ambiance du projet avec nos désormais connus personnages - faudra s'efforcer de leur trouver un nom.

Concernant l'aspect technique, le site utilise bien évidemment le XHTML 1.1 et le CSS 2.0, tout en respectant les normes éditées par le W3C. Nous nous sommes servis aussi du PHP pour faciliter la génération des pages web. Les news sont gérées en MySQL. le reste du site ne demandant pas une mise à jour fréquente, ce serait un gaspillage de ressources d'utiliser MySQL pour l'affichage de liens par exemple.

Le menu, qui était d'abord en HTML statique, a été refait en flash, afin d'augmenter le sentiment de convivialité et le dynamisme du site.

Avant de rentrer plus en détails sur chaque partie du site, il est continuellement consultable à cette adresse : http ://webdgp.free.fr/ - bonne visite !

News

La section "News" permet de rapidement se tenir au courant des dernières news concernant le site et le projet, elles sont classées par date de manière à toujours maintenir l'actualité en haut de la page.

DGP

Cette seconde partie permet de mieux situer le projet pour l'utilisateur. Le type de projet et le ton sont donnés via un long texte, qui était d'ailleurs le texte d'introduction du cahier des charges. Les captures d'écran viendront s'y ajouter dès qu'elles seront suffisament significatives pour le visiteur.

Fichiers

Afin de permettre aux visiteurs, et futurs INFOSUP de contempler (ou pas) notre travail, la section “Fichiers” accueille nos différents rapports au

format PDF, auxquels viendront s'ajouter les exécutables et sans doute quelques bonus si le temps nous le permet.

Team

La partie "Team" sert à mettre en avant les différentes personnes (bon ok on est que deux) qui s'investissent dans le projet. Des informations viendront peut-être s'ajouter, si l'inspiration nous vient (et nous connaissant, ça va pas être triste).

Liens

La section "Liens" présente différents liens, classés selons différentes catégories et classés par ordre alphabétique de contenu. On y trouve des liens

consacrés à la programmation Pascal Object dans une première section intitulée “Delphi”. Ensuite, nous avons mis quelques liens en relation avec Direct X. La section “Divers” comprend les liens ne pouvant être classés ailleurs. Et enfin, la section “Autres projets” contient des liens vers d’autres projets de la promotion, cela va de soit. Les liens seront ajoutés au fur et à mesure de leur découverte.

Vide

La section vide, quant à elle... est vide.

Conclusion

Le site du projet atteint son objectif : coller au plus près à l’esprit du projet tout en restant un lieu de présentation et d’information.

8 Finalisation

Réalisation : Laurent

Période : Juin

Prérequis :

- Validation du module de travaux manuels avancés
- Maîtrise du principe d'*autorun*

Points forts :

- Identité visuelle
- Compatibilité entre différents ordinateurs
- Améliore la prise en main par l'utilisateur

Difficultés rencontrées :

- La précision requise pour les reliures, le boîtier...

Introduction

La finalisation du projet était aussi importante que le projet en lui-même. On peut faire un logiciel ultra performant, la communication visuelle lui assure une partie de son succès. Cette finalisation implique plusieurs domaines : rédaction de manuels, design du boîtier, production d'une interface d'installation et organisation du CD.

Les manuels

Puisque ceux-ci sont en principe destinés à des personnes n'ayant pas forcément des connaissances poussées dans le domaine de l'informatique, les manuels se devaient d'être simples et précis. Ce n'est pas la rédaction qui fut la plus douloureuse, mais la reliure en ce qui concerne le mode d'emploi. De plus, comme l'ensemble des éléments accompagnant notre exécutable, ces manuels sont en harmonie avec l'ambiance générale se dégageant du projet.

Le boîtier

Pour l'élaboration du "package" il a tout d'abord fallu se munir d'une boîte DVD neuve ainsi que d'une feuille de papier adhésif pour la jaquette du CD. L'apparence extérieure du boîtier a nécessité une brève observation de boîtes de jeux commercialisés, le plus dur étant de faire des jaquettes à la bonne taille. Une fois le tout mis en place, l'apparence du boîtier était exactement celle que nous souhaitions. Et un beau "package", ça poutre !*

L'interface d'installation

Le fichier d'installation a été réalisé avec "Inno Setup". Ce logiciel permet de rédiger un script qui une fois interprêté va permettre la compilation du fichier d'installation. On peut ainsi choisir les raccourcis à mettre dans le menu démarrer, la langue du fichier d'installation, l'icône utilisée .... Ce logiciel demande un bref temps d'adaptation mais une fois le langage scripté pris en main, Inno Setup se révéle être un outil relativement puissant.

Le CD

Le CD a nécessité la création d'un autorun. Lorsqu'on insére le CD dans un lecteur CD-Rom, si le lancement automatique n'est pas désactivé, un menu apparaît permettant : le lancement de la procédure d'installation, le parcours du CD (pour jeter un oeil aux sources, ou aux documents fournis), le lancement du site web ou la fermeture du menu. Cet élément participe à la prise en main destinée à des utilisateurs non avertis - hé oui ça existe encore.

Conclusion

Les finalisations se sont avérées être indispensable mais plutôt agréables à effectuer. Lorsque l'on voit le résultat final, on ne regrette pas d'y avoir passé un certain temps.

IV Récit de la réalisation

Chroniques d'un projet acronymique

Bien l'bonjour vous tous. Si vous lisez ça, c'est qu'on a fini, qu'on y a survécu, et que quelque part, on en redemande. Et pour couronner le tout, il m'a été donné le plaisir de faire un rapport un peu plus vivant que les centaines de lignes de codes qui donneront à certain une moue désapprobatrice, à d'autre un air un peu dépressif. Voici donc, avec un peu de différé, la vie mouvementée d'une poignée d'étudiants presqu'en marge de la société.

Introduction

DGP, ca commence avec deux colocs qui se connaissent, un redoublant motivé plein d'expérience, et un inconnu, mais hey, il fait bonne impression lui. Ni une ni deux, des mains sont serrées, des ébauches de jeu griffonnées, on trouve un nom de team, Hexen, ça poutre comme disait l'autre, bref, fin de séminaire, la machine est en route.

Soutenance 1 : Les sanglots longs...

D'Octobre à Décembre, pas trop de prise de tête. Fait moche dehors, pis on est pas habitués à la ville, et s'mettre en route dans une nouvelle école avec plein d'maths, c'est pas si simple. Pourtant, le projet reste en tête. On trouve une organisation préliminaire, des idées à placer dans le cahier des charges, on se fait une idée de la charge de travail à distribuer, on jette des ébauches de code sur le papier. Et puis on discute, les anciens sont là, guident, répondent aux questions stupides en pointant la direction dans laquelle chercher, bref, petit à petit, on commence à voir ce que l'on a à faire en six mois.

Décembre. Aie. La première soutenance arrive, pas de jeu sous la main. On n'a que nos essais divers, en structurant un peu ca devrait passer. Mais première surprise, Nicolas, après un an et demi d'Epita, jette l'éponge et va se trouver un endroit plus facile à vivre. Dans un coin de notre esprit, on l'envie tous un peu.

Janvier, soutenance. Eck, ne nous vîmes-nous pas plus que trois ? Pas

grave, zéro c'est critique, au dessus il reste une chance. D'ailleurs, pour une première fois, la redoutée soutenance de projet se passe plutôt bien. On a du code perso, les tutos lus ont été compris et assimilés, pas de triche pour cette fois. Bon, on affiche que des boules noires sur fond bleu, mais l'octree est codé, on sait plus ou moins se servir de DirectX, place donc, au jeu lui même.

Soutenance 2 : A la Sainte Agathe...

Fini de rigoler. Ouais, même vous dans le fond là. On bosse. Et dur. Enfin presque tous. La courte période entre les deux premières soutenances est difficile. Colles de maths tout ça, dur dur de bosser entre les vacances, mais on se force un peu.

Vacances de février. "Ouf !" général. Du moins partout ailleurs. Ici, la majorité les passe à coder, débugger, se prendre la tête et du café, until jour = J. L'reste d'l'équipe euh, les passe on-ne-sait-où, probablement à se demander ce qu'il fait dans st'école. Avec décence, Alexandre ne se présentera pas à la soutenance, et assumera n'avoir rien accompli qui fasse avancer le projet.

Jeudi sans doute, 4 :16am. Les yeux bouffis, je relis le rapport de soutenance 2. Quelques fautes qui trainent, ça sent l'urgence de dernière minute, c'est vrai que s'y prendre à l'avance ç'aurait été bien mieux, mais on est là pour apprendre de nos erreurs après tout. Bref, soutenance rigolote, plein de petites boules rouges et vertes qui rebondissent dans un cube, c'est loin d'être un FPS, mais on s'en sort. Du moins cette fois.

Soutenance 3 : Mouack.

Ca ne pouvait pas pas finir comme ça, pas si tôt, pas après tout l'acharnement qu'on y a mis, pas après les nuits de vacances passées à coder, pas lorsqu'on a su garder la tête hors de l'eau dans d'autres matières ma fois plus importantes que struc, non, ça pouvais pas...

Et heureusement d'ailleurs, eh. Mars Avril, renforts incoming, Renaud arrive, et tout de suite, ca poutre plus mieux. Ack, évidemment, toujours les mêmes déboires de pointeur de merde que-j'avais-pas-déréférencé, bref,

quatre jours à patauger, heureusement, les autres avancent, on se soutient moralement, et on arrive à quelque chose. (Note to self : prévoir quand même de la marge la prochaine fois)

Et donc, soutenance trois, samedi, midi quarante-cinq, la classe, dernier groupe à passer ou presque, on cherche à pas trop lasser, rapport trop court mais optimiste, présentation viva voce qui marche et tout, nous, on se sent las, mais fiers d'avoir tenu le coup.

Soutenance 4 : Je rêvais d'un autre monde...

Vint le temps du regard en arrière. Le trajet parcouru est conséquent. Jamais une année scolaire n'avait apporté autant niveau technique. Par contre, la sociabilité en a souffert, mais bon, on en a presque plus rien à foutre. La lassitude s'installe peu à peu. C'est le ras-le-bol général. On en a marre de ce projet, plus envie de continuer, la flemme, on codera tout après les partiels. D'un autre côté, le projet ne reste qu'un projet, partiels réussis, passage en spé, mais trop de temps sur le projet risque de donner à l'été un arrière goût de matrices et d'EVs.

Début Juin, il est temps de s'y remettre. C'est dur, très dur, on sent la tension monter, parfois ça craque, mais le calme revient. Tant bien que mal, on est bien loin de ce que l'on espérait au départ, mais le tout se tient. Reste à paufiner le rapport de projet et autres publications. Une bonne matinée de sommeil, et on l'espère, tout se passera bien. D'ailleurs j'frais mieux d'y aller, l'jour se lève...

V Impressions

1 Laurent

Ce projet aura été enrichissant sur de nombreux points : travail en équipe, programmation, présentation d'un projet...En tant que chef de projet il m'aura appris à coordonner les différents membres et éléments. Ce fut tout un travail d'organisation. Préparer les rapports, les soutenances, me lever de bonne heure pour aller relier les rapports, préparer le package final...La réalisation du projet m'a entraîné dans différents domaines de sorte que j'ai un peu touché à tout : de la 3D aux contrôles, du son au site web, de la réalisation du package au parseur de maps et bien d'autres choses encore...Ce projet me donne envie de creuser encore plus l'année prochaine...

En ce qui concerne la partie graphisme ; je regrette l'absence de sources pour DirectX sur internet. En effet, de nombreuses sources pour Delphix sont disponibles mais celles pour DirectX pur sont réellements inexistantes. Un de mes ACD, Matthieu Bucchianeri, m'aura bien aidé dans ce domaine. Mes connaissances m'ont en outre permis de regarder des sources codées en C++ et de les transposer en Pascal. Je regrette de ne pas avoir eu le temps de paufiner le moteur graphique, bien que le résultat final soit assez joli. Certains bugs de la librairie DirectX pour Delphi ont aussi été assez frustrants et un seul n'a pas pu être résolu, celui des sprites.

Un de mes rares regrets concernant ce projet est de n'avoir pas pu voir l'équipe rester entière. L'abandon d'un membre au bout de deux mois, la passivité d'un autre pendant six mois ; tout cela a fort pesé sur la réalisation du projet, et l'on en a un peu payé le prix pour cette dernière soutenance. Heureusement que Renaud nous a rejoint. Il n'a pas été facile pour lui de s'intégrer en cours d'année dans ce projet qu'il ne connaissait pas mais il a plutôt bien réussi. Il a réellement bossé pour réaliser ce que je lui avais demandé - réseau, son - et il a même participé au final à la création de la map présentée lors de la soutenance. Si sa participation au projet peut sembler peu conséquente comparée aux nôtres, il convient de se rappeler qu'il est arrivé juste avant la troisième soutenance, et que pour le temps qu'il a passé avec nous, il ne s'est pas tourné les pouces ! Au final l'ambiance au sein du groupe est restée bonne du début à la fin du projet malgré quelques accrochages, mais c'est la vie en société !

Pour conclure ce projet me laisse un peu sur ma faim. Même si très avancé il n'est pas tout à fait fini. J'attends avec impatience l'année prochaine pour me relancer dans cette aventure et faire encore mieux...

2 Florent

Mouack. Pour ma part, j'ai commencé ce projet des idées plein la tête, et avec beaucoup de motivation. Le simple fait d'avoir à réfléchir à une manière d'implémenter dans le projet ce que l'on a envie d'y inclure fait prendre conscience que peut-être, tout ne sera pas aussi simple que ce que l'on espère. Mon rôle ici a été bien plus technique que relationnel. J'ai pris en charge la programmation globale du projet, l'architecture et l'organisation de ses éléments. Ainsi donc, le DGP m'a surtout marqué par une abondance de code, de recherches algorithmiques, et de maux de tête. Non que je regrette quoi que ce soit. L'expérience que j'ai pu y retirer au niveau de l'organisation technique est immense et sera, je l'espère, durable.

Mis à part la programmation pure et dure, la réalisation pour la première fois d'un projet important par équipe m'a fait prendre conscience des qualités d'expression, de coopération et d'ouverture d'esprit qui je pense, vont m'être demandées par la suite. De plus, ce projet a permis dans une certaine mesure de développer des aspects créatifs de ma personnalité, notamment dans l'écriture ou la réalisation d'illustrations intégrées dans les diverses DG-Publications. Eck.. un peu hors limites parfois, mais si cela ne peut servir à l'ingénieur, on peut supposer que cela se démontre d'une certaine utilité pour l'individu en tant que membre d'une société de vie communément qualifiée de sociale (ouais bref, on peut ici comprendre que certains se posent des questions).

Chuis tout trist' que ça s'arrète ici. M'enfin, sûr que j'n'ai ras-le-bol, comme un peu tout le monde, mais voir la quasi-seule occasion d'ma vie d'créer un jeu vidéo partir comme ça... Cha m'rend tout chose. Dommage d'ailleurs que l'on ait pas pu maintenir une équipe forte et soudée tout au long de l'année. Ce sont des choses qui arrivent, et souvent, me direz vous ; et à dire vrai, faudra plus pour me rendre dépressif. Et puis quelque part dans notre désorganisation chaotique a surgi une sorte d'instinct, qui fait que le trio de fin a plutôt bien marché.

J'va soon d'voir rendre el' papier lô, so... Bref, expérience enrichissante, nuitées ahurissantes, assiettes de pâtes trop nourissantes... La vie d'épitéen, d'un côté j'accroche, d'lautre... Hey, j'ai bien l'temps d'voir venir.

3 Renaud

Me concernant, ayant rejoint le groupe seulement à partir de la troisieme soutenance, j'ai dû m'adapter à l'architecture du projet qui m'était jusqu'alors inconnue. L'Ambiance dans ce groupe est bien meilleure que celle qui pouvait exister dans le "groupe" dans lequel j'étais auparavant et dans lequel je semblais être le seul concerné... C'était donc fort agréable de travailler avec Florent et Laurent qui n'ont pas peur du travail et qui étaient trés motivés, bien qu'au départ je me sentais un peu dépassé. Heureusement en plus de toutes les qualités précédemment citées, ils sont gentils :). Bon, j'en rajoute peut être un peu ...

Je n'ai traité que des parties annexes du projet, les deux grandes parties, à savoir le moteur graphique et le moteur physique étant pour leur part dèjà bien entamées. Ceci ne m'a pas empêché d'apprendre des notions et des techniques qui m'étaient jusqu'alors étrangères. Comme la programmation objet, l'utilisation de différentes bibliothèques (non non, je n'ai pas dit librairies) comme FMod pour le son, très sympa à utiliser ; ou ICS pour le réseau que je n'ai au final pas utilisé (une vrai usine a gaz !) mais dont les tutoriaux m'ont permis de découvrir des astuces et de comprendre le fonctionnement d'un réseau.

J'ai aussi pu toucher à la partie graphique, avec les menus. Et à la 3D avec la réalisation d'un terrain presque entièrement à la main n'ayant pas eu le temps de réaliser un éditeur. J'en ai d'ailleurs profité pour revoir les bases de ma géometrie dans l'espace.

Au final,les deux gros points que j'ai retenus du projet sur l'année en général, c'est qu'il faut bien choisir ses collègues et ne pas avoir peur d'y passer ses vacances. C'était une très bonne experience technique, humaine (les délires... parfois mêmes grossiers hum ! n'est ce pas Laurent ?) et physique (ne pas dormir, ne carburer qu'aux pâtes, transporter un PC fixe dans le metro...). Voilà, c'est a peu près tout. Ah ! j'oubliais : GNIIIIIIIIII !

VI Conclusion

C'est donc à moi que revient le mot de la fin - mais vous ne saurez jamais qui je suis hihi. Que dire de plus quand tout a été dit au travers de ce rapport ? L'erreur à ne pas commettre serait de dire que nous sommes contents de notre travail.

Le projet aura été une expérience fort enrichissante. Même si tout au long de notre scolarité nous avions déjà effectué des travaux en groupe, rien ne rivalise avec ce que nous avons vécu cette année. De plus, bien que nous en ayons appris beaucoup sur le travail en équipe nous avons aussi beaucoup appris sur la façon de travailler ensemble.

Que dire aussi au sujet de la programmation ? Il n'y a pas mieux que cette expérience pour faire nos premiers "vrais" pas dans le monde du développement informatique. C'est dans cette réalisation que les cours d'algo prennent tout leur sens, et ne restent plus de simples ensembles théoriques.

Dans ce cas là, pourquoi ne pas conclure sur une note positive du style "nous sommes fiers de vous présenter ce que nous avons mis un an à réaliser" ?

"*I can't get noooo satisfaction*" ; ou presque. Bien que déjà très poussé, nous aurions pu faire mieux et nous en sommes conscients. Aussi plutôt que de nous contenter de ce premier projet, nous profiterons de nos vacances avec le futur souhait de faire encore mieux l'année prochaine, et encore mieux tout au long de notre carrière de sorte que la programmation - loin d'être un simple exercice - devienne un vrai challenge qui nous poussera chaque fois un peu plus loin...