

Stardust

Rapport première soutenance

Koin-Koin z'Team

CRESTEY BENOÎT
CALVET BRUNO
CARRIGNON DAVID
EGGENSPIELER RÉMI

INFOSUP

6 janvier 2003

Table des matières

1	Introduction	3
1.1	Présentation	3
1.2	Présentation du groupe	3
1.3	répartition des tâches	4
1.4	Précision sur le fonctionnement général du jeu	4
2	Avancement du projet	5
2.1	Moteur Graphique	5
2.1.1	Moteur OPENGL	5
2.1.2	Moteur de déplacement	5
2.1.3	Importateur de fichier 3DSMAX	6
2.1.4	Moteur 3D	7
2.1.5	Création de modèle 3D	7
2.2	Moteur de texte	8
2.2.1	Principe	8
2.2.2	Réalisation	9
2.2.3	Conclusion	9
2.3	Acquisition du clavier	10
2.3.1	Documentation	10
2.3.2	Principe de fonctionnement	10
2.3.3	Conclusion sur les contrôleurs	11
2.4	Principe des scénarios	12
2.4.1	Problématique	12
2.4.2	Principe	12
2.4.3	Réalisation de l'éditeur de scénario	13
3	Avance du projet	14
4	Ressources	15
5	Prochaine Soutenance	16
6	Conclusion	16
7	Annexes	17

1 Introduction

1.1 Présentation

Notre groupe est composé de Benoît CRESTEY (SUP B2) le chef du groupe, David CARRIGNON (SUP A2), Rémi EGGENSPIELER (SUP D2), Bruno CALVET (SUP B1).

Notre est un jeu de simulation spatiale en 3D jouable en solo et étant inspiré de Starlancer, Tachyon et quelques autres très bon jeux de ce type.

Pour cette première soutenance nous devons présenter une ébauche du moteur graphique, une ébauche de l'éditeur d'évènement, la gestion du clavier ainsi que nos diverses documentations accompagnant ces diverses parties.

1.2 Présentation du groupe

Nous allons vous présenter un peu plus précisément les membres de la KOIN KOIN'Z TEAM :

- Benoît Crestey, dit Multivac est le chef tyrannique de la team. Sa passion oscille entre son écran 22 pouces et le fût de bière qu'il a acheté la veille. Sa musique "la danse des canards" et son crédo "Koin-Koin!!!" font de lui un amoureux de ces sympathiques palmipèdes, spécialisé dans l'ouverture de la "Leffe" au disque dur (véridique!).
- David Carrignon, dit k-riboo est l'Expert en quaternion et autres réjouissances vectorielles. Sa passion passant de sa table de muscu à l'entretien de ses abdos kronenbourg, et son crédo "Gni? Ca marche pas...." font de lui un être tout à fait pittoresque dans le groupe.
- Rémi Eggenpieler, dit Pimouf est le parisien comique du groupe se faisant à loisir enfermer tout seul sur le balcon par -20C car il ne peut s'empêcher de fumer sa clope toutes les 20 min 18.
- Bruno Calvet, dit Almiriad est l'auvergnat bouseux du groupe. Maîtrisant comme un chef l'art culinaire, entre pâtes à la crème et autres réjouissances trois étoiles, il est aussi celui qui chante le plus de chansons paillardes à la minute. son crédo " "

1.3 répartition des tâches

Voici une courte liste des différentes tâches de chacun pour cette soutenance :

- Crestey Benoît : Éditeur d'évènements
- Carrignon David : Ébauche du moteur graphique ainsi que du loader
- Eggenspieler Rémi : Moteur d'affichage du texte
- Calvet Bruno : Gestion du clavier

1.4 Précision sur le fonctionnement général du jeu

Le projet Stardust sera composé d'un moteur d'affichage (ou moteur graphique), ainsi que d'un moteur de gestion, qui est chargé de prendre en charge la gestion des données, c'est à dire les mettre en place (initialisation des listes chaînées) et les gérer. Ce moteur de gestion appellera les différents modules composant le jeu et mettra en relation les données de ceux-ci.

En simplifiant, une fois le jeu lancé, le processus sera celui-ci :

- Acquisition des contrôles claviers, souris ou Joystick
- Traitement des événements
- Gestion des vaisseaux
- Gestion de l'intelligence artificielle
- Gestion des déplacement des vaisseaux
- Gestion des Tirs, missiles
- Gestion des collisions
- Affichage de la scène

2 Avancement du projet

2.1 Moteur Graphique

Pour la première soutenance, il était prévu que l'on réalise une ébauche du moteur graphique. Nous avons ainsi commencé par créer un début de moteur graphique sous OpenGL, qui nous permettait d'afficher des objets comme des triangles ou des carrés. Ensuite, il fallait créer un déplacement en 3D. Pour cela, nous avons utilisé des quaternions. L'étape suivante fut de réaliser un loader de fichier 3DSMax pour charger des objets dans la mémoire. Nous avons décidé à ce moment précis de l'ébauche du moteur, de changer d'API graphique. Pour des soucis d'optimisations futures, nous avons ainsi préféré utiliser Direct3D.

2.1.1 Moteur OPENGL

Dans un premier temps, il fallut se renseigner sur OpenGL. Internet fut une grande source de tutoriaux traitant d'OPENGL qui nous permit de comprendre rapidement comment l'utiliser. Nous avons aussi utilisé le livre "OpenGL 1.2 de Campus-Press". De plus, l'un des membres du groupes avait déjà utilisé cette API graphique pour créer un moteur 3D lors de son précédent projet.

Ainsi nous avons pu réaliser sans trop de problèmes une ébauche du moteur. Une fois notre premier objet créé, nous avons essayé de le faire se déplacer comme un vaisseau.

2.1.2 Moteur de déplacement

Pour créer un moteur de déplacement, nous avons d'abord conçu celui du joueur. Suivant les touches pressées du clavier, l'objet 3D représentant le joueur (en l'occurrence un prisme) devait être déplacé dans l'univers. Rapidement, un problème se posa : Comment arriver à effectuer des rotations 3D correctes ? En utilisant les rotations classiques d'OpenGL, nous ne pouvions pas faire tourner le vaisseau du joueur de façon correcte.

Ceci est dû aux commandes de rotations d'OpenGL qui s'effectuent à partir du repère d'origine, alors que les rotations du vaisseau doivent être effectuées autour du repère du vaisseau. Prenons par exemple, un croiseur placé quelque part devant la caméra, incliné de 45° sur le côté gauche. Si le joueur presse la touche du bas, le nez du vaisseau doit pivoter de quelques degrés vers le haut par rapport à son repère. Or, à l'écran, il va pivoter vers le haut et la gauche en même temps, ce qui est normal. Donc, dans ce cas de figure, la rotation unique du vaisseau autour de son repère est équivalente à deux rotations combinées autour du repère d'origine. Et comme nous ne pouvons effectuer que des rotations de ce deuxième type, nous avons dû chercher une solution. Nous l'avons trouvé dans les quaternions.

Ces objets mathématiques très puissants sont utiles pour la résolution de nos problèmes de calculs. Le site *www.game-dev.net* contient plusieurs tutoriaux sur l'utilisation des quaternions. Ce sont en fait des vecteurs 3D munis d'un angle autour de lui-même. Le petit plus des quaternions, c'est l'optimisation car il existe d'autres méthodes plus lourdes. Ainsi, chaque objet 3D est associé à un quaternion qui permet d'obtenir les 3 rotations par rapport à l'origine, ce qui permet de le positionner dans le repère monde. Pour appliquer une rotation locale à un objet, il faut simplement calculer un quaternion temporaire à partir de l'angle et de l'axe de rotation. Ensuite, on multiplie le quaternion temporaire avec celui de l'objet, et on obtient celui qui représente la nouvelle rotation de l'objet.

Il ne reste plus qu'à afficher le tout pour constater que cela résoud nos anciens problèmes. Une fois ceci terminé, il fallait créer le déplacement du vaisseau. Nous avons créé pour cela une fonction calculant le vecteur direction du vaisseau à partir de son quaternion. Ainsi, on peut calculer son vecteur accélération et obtenir sa nouvelle position. Dès que le déplacement fonctionna pour le joueur, il ne resta plus qu'à créer une fonction générale. Une fois le moteur de déplacement totalement terminé, nous avons remplacé le prisme par un vaisseau créé sous 3DSMax.

2.1.3 Importateur de fichier 3DSMAX

Nous avons dû prendre une décision importante : c'est à dire choisir entre faire un importateur de fichier ASE ou 3DS. Chacun a ses avantages et ses inconvénients. Les fichiers ASE sont de type texte, donc ils sont facilement modifiables et compréhensibles mais ils sont gourmands en mémoire. Par conséquent, ils seraient long à charger pour notre jeux, alors que les fichiers .3DS sont de type binaire. Ce qui rend ces fichiers difficiles à charger car ils sont incompréhensibles, mais leur taille est beaucoup plus petite. De plus, les tutoriaux sur ces fichiers sont beaucoup plus rares.

Nous avons donc choisi de charger des fichiers 3DS pour leurs avantages. Nous avons trouvé quelques sites internet contenant des tutoriaux pour charger ces données. Cependant, il fallut faire le tri car certains expliquaient comment lire les données en parcourant les fichiers de façon récursive, ce qui n'est pas très propre. En combinant les méthodes de plusieurs de ces tutoriaux, nous avons réussi à créer un importateur capable de charger plusieurs objets contenus dans un seul fichier. Tout les objets sont stockés en mémoire et peuvent être affichés par une simple procédure. Pour chaque objet on stocke les coordonnées de position, les normales et les coordonnées de textures de chaque sommet, ainsi que les numéros des sommets composant chacune des faces. Pour le moment, les textures ne sont pas chargées par l'importateur ; elles sont chargées à la main dans le code principal. Une fois toutes les données chargées, il faut recalculer les normales de chaque sommet pour obtenir des normales appliquées au faces.

2.1.4 Moteur 3D

Dans le soucis de meilleures performances, nous avons décidé de changer d'API graphique. Comme nous utilisions OpenGL 1.2 qui a plus de 4 ans, il était logique que Direct3D 8.1 soit plus rapide car il a moins de 2 ans, et utilise les dernières technologies graphiques. Il utilise des vertex et des index buffer ce qui diminue les calculs sur les sommets. Plus précisément, le vertex buffer est un tableau stocké dans la mémoire de la carte graphique où tous les sommets des objets sont stockés. L'index buffer est un tableau aussi stocké au même endroit mais il contient l'indice (ou numéro) des sommets composant chaque face. Pour afficher un objet, direct3D prend donc les indices par 3 et crée chaque face. L'intérêt de ce tableau est de ne pas répéter plusieurs fois les mêmes calculs sur les sommets utilisés plus d'une fois. Cette technique de stockage des données accélère beaucoup la vitesse de rendu par rapport à OpenGL. Lors de nos tests, au niveau des performances, notre projet a gagné 50 % d'images par seconde en utilisant Direct3D au lieu d'OpenGL (130 images par seconde au lieu de 80).

Pour le moment, notre jeu tourne dans une résolution de 1024x768, il affiche du texte, un vaisseau et une sphère englobant la scène pour représenter l'univers. Différentes vues de caméra seront gérées. Mais la caméra dans le vaisseau du joueur n'est pas encore totalement terminée. Pour le moment, les textures utilisées sont de grande dimension, mais il est prévu que leur taille varie en fonction des options d'affichage choisies par le joueur.

Les lumières sont aussi gérées par notre moteur et nous pourrons ainsi créer des effets de lumières lors des diverses explosions.

2.1.5 Création de modèle 3D

Pour créer les modèles 3D de notre jeu, nous utilisons le logiciel Discreet 3D Studio Max 5. Nous avons choisi ce logiciel pour les possibilités qu'il nous offre. Pour apprendre l'utilisation de ce logiciel nous nous servons du livre 3D Studio Max 3 Campus-Press et d'un magazine mensuel studio multimedia sur la création numérique. Nous avons dû apprendre des techniques de modélisations complexes pour créer des modèles 3D de moins de 500 polygones dans le soucis des performances, car notre moteur ne pourra pas garder une certaine fluidité si il doit afficher trop de polygones. Notre premier vaisseau fut créé à partir de lignes courbes formées de 2 ou 3 segments, ce qui nous permet de contrôler la complexité de notre modèle. Cette technique de construction nous a semblé la meilleure manière de réaliser nos modèles. Par ailleurs notre importateur n'est pas capable de charger des modèles comportant plusieurs textures. Nous devons donc réaliser des textures complexes pour que nos créations ressemblent à quelque chose. L'utilisation de logiciels tels que Deep Paint 3D pourra nous aider pour la création de nos textures.

2.2 Moteur de texte

Pour la première soutenance, nous nous sommes imposé de pouvoir écrire du texte à partir du moteur graphique cité plus haut. Il fut convenu, de premier abord, de ne pas utiliser DirectDraw en post-rendu de l'image pour pouvoir afficher le texte ; mais bien d'utiliser le moteur graphique lui même.

Le premier cas pratique qui s'imposait à nous fut de rendre la scène 3D, puis de faire un post-rendu avec DirectDraw permettant l'affichage d'images en transparence au dessus de la scène, permettant ainsi l'affichage aisé de textes et autres effets spéciaux. Mais en y réfléchissant bien, nous avons compris que la manipulation d'images à travers DirectDraw risquait de faire chuter les performances globales de rendu d'image ; en effet il aurait alors été question de la création d'un canevas DirectDraw au dessus de l'image rendu, puis du chargement et de la manipulation de bitmaps, puis d'un deuxième rendu de l'image. On comprend tout de suite que cette solution, bien que simple à mettre en place, se révèle très coûteuse au niveau vitesse d'affichage.

Il fut alors décidé d'afficher le texte au moyen du moteur graphique élaboré au même moment. En effet, cette solution permet d'afficher les objets graphiques par le moteur 3D. Elle repose sur le principe de la création de rectangles dans la scène et d'y plaquer les textures des graphiques désirés. Bien qu'augmentant le nombre de polygones de la scène, cette technique se montre bien plus rapide au niveau du rendu et de l'affichage, nous faisant gagner de précieuses images rendues par seconde. Bien que moins coûteuse, cette solution se révèle par contre difficile à mettre en place, et pose plus un problème de logique que de performances .

2.2.1 Principe

Le principe de cette technique se limite pour l'instant à l'affichage de texte dans la scène, qui pourra être utilisé ensuite pour indiquer la vitesse du vaisseau, les messages passés dans le jeu, l'élaboration du menu, affichage titre en temps réel, etc... En voyant plus loin, on se rend compte que l'on sera capable d'effectuer des effets spéciaux sur les textes, permettant de les animer, et effectuer des rotations sans les effets indésirables de l'animation 2D simple. Le principe repose sur le fait d'arriver à découper une texture contenant les différents caractères, et de plaquer le morceau de texture sur un rectangle aux dimensions du morceau de texture, et ensuite d'arriver à les aligner pour former des mots, puis des phrases. Ainsi fait, il ne reste plus qu'à l'afficher à l'endroit voulu devant la caméra.

2.2.2 Réalisation

Une texture contenant les différents caractères de la police fut créée. Il fallut alors déterminer les coordonnées de chaque lettre dans la texture. Une liste des coordonnées fut éditée pour chaque lettre, et un tableau fut créé, contenant toutes les informations utiles. Des procédures de calcul ont été mises en place permettant, à partir de deux points significatifs, de calculer les coordonnées des deux autres points du rectangle de texture contenant la lettre, puis de déterminer sa longueur et sa largeur. La première étape fut d'arriver à appliquer le morceau de la texture déterminé par ses coordonnées sur un carré de taille quelconque. Malgré une certaine déformation minimale, due à la différence de taille des morceaux de textures, le but fut atteint.

La deuxième étape fut d'arriver à adapter la taille du rectangle à partir de la longueur et la largeur de la texture du caractère. La mise en place fut un peu plus délicate mais fut réussie.

La troisième étape fut d'arriver à faire une procédure qui prend en paramètre une chaîne de caractères, qui la parcourt et qui cherche à chaque caractère de la chaîne les coordonnées correspondantes dans le tableau de données. Il fallait aussi créer une procédure qui créait un rectangle pour chaque caractère de la chaîne. La tâche fut dure, et plus spécialement au niveau de l'espacement entre chaque caractère ; en effet, au début nous nous sommes trompés car nous affichions d'abord le rectangle texturé puis nous calculions l'espacement avec le caractère d'avant. Erreur minimale dans l'algorithme, mais qui donnait des choses étranges en résultat. Ceci fait, la partie affichage de texte était bientôt terminée, et après quelques petites corrections, le texte s'alignait parfaitement, espacements compris. Il ne manquait alors plus qu'à aligner le texte en haut à gauche de la caméra, pour pouvoir en faire un compteur de vitesse.

2.2.3 Conclusion

Malgré la mise en place d'un principe assez complexe, par rapport à un affichage `DirectDraw`, il est maintenant facile d'améliorer l'algorithme, et de le rendre générique pour des textures de polices que nous aurons personnalisées, et ainsi faire un affichage de texte avec différentes polices.

2.3 Acquisition du clavier

Pour cette partie nous avons immédiatement choisi d'utiliser DirectInput car cela nous semblait le moyen le plus aisé de gérer le clavier et plus tard le joystick. Gérer la souris ne nous a pas paru intéressant car celle-ci, bien que pratique pour le mouvement, devrait être combinée au clavier et il y aurait beaucoup de commandes à gérer d'une seule main pour le joueur, l'autre étant occupée par la souris.

Pour le moment le joystick n'est pas géré car nous n'en avons pas en état de marche à notre disposition.

2.3.1 Documentation

Nous avons tout d'abord fait des recherches sur des sites internet parlant de DirectInput, de ses intérêts, ses inconvénient.

Nous avons également utilisé des livres pour nous informer de la méthode de fonctionnement de DirectInput.

Une fois ces recherches terminées nous avons pu commencer à intégrer les fonctions de directInput a notre jeu. DirectInput est très pratique et permet d'utiliser un buffer pour voir l'état des touches du clavier et ainsi de pouvoir stocker les différentes touches appuyées en même temps car le type de notre jeu est un type où en moyenne il y a 4 touches enfoncées en même temps et l'utilisation d'un buffer était la manière la plus simple de gérer les entrées clavier.

2.3.2 Principe de fonctionnement

Nous avons décidé pour cette première soutenance de faire deux choses différentes concernant le clavier :

- Une partie intégrée au moteur graphique qui utilise DirectInput pour le déplacement du joueur
- Un petit logiciel sous windows permettant de configurer les touches comme on veut qu'on peut sauvegarder et ensuite charger.

Pour faire la partie en DirectInput intégrée au jeu, il faut tester si le joueur a bien la bonne version de DirectX et si il a bien un clavier de branché a son ordinateur. Une fois ceci testé, la commande de d'acquisition des touches est comprise dans la boucle du jeu, ainsi à chaque tour de boucle le programme met dans un buffer l'état de chaque touche du clavier et ensuite on teste pour chaque touche utile si elle est enfoncée, alors celle-ci provoque une action dans le jeu. Pour sortir de la boucle de jeu la fonction traitant les actions du clavier renvoie un booléen lorsqu'on appuie sur la touche "ESCAPE" qui fait se terminer la boucle de jeu et quitte le jeu.

Nous avons aussi fait un petit logiciel permettant de configurer ses touches comme on le souhaite ce qui existe dans quasiment tous les jeux actuels et est indispensable pour un jeu tel que le notre.

Ce programme fonctionne sous windows et permet de choisir chaque touche pour chaque commande. Pour ce faire il suffit de placer le curseur dans la case située à côté de la commande désirée et d'appuyer sur la touche requise, on verra alors s'afficher le nom de la touche dans la case, si cette touche est déjà affectée rien n'apparaîtra. Deux boutons existent le premier est nommé "RESET" et permet de remettre toutes les valeurs à zéro si le joueur veut complètement changer les paramètres, le deuxième bouton est nommé "TEST" et permet de tester les touches paramétrées afin de voir si les touches correspondent bien à ce qui a été défini.

Le logiciel permet aussi de sauvegarder les paramètres, pour ce faire il suffit de cliquer sur "Sauver" dans le menu "Fichier", le programme sauvera alors la touche pour chaque commande dans un fichier selon la syntaxe suivante : '\$la commande'|'nom de la touche'. L'option "Charger" dans le menu permet de récupérer la dernière configuration sauvegardée à partir du fichier de sauvegarde, pour ce faire le programme parcourt le fichier de sauvegarde et stocke le paramètre dans un tableau, la case contenant une commande précédant la case contenant la touche associée. Au lancement du programme le chargement est fait automatiquement afin que le joueur puisse récupérer ses derniers paramètres enregistrés.

2.3.3 Conclusion sur les contrôleurs

Pour cette soutenance la gestion du clavier est terminée comme prévu car tout le système de touches est géré il reste juste à intégrer la configuration du clavier dans l'interface graphique du jeu afin que le joueur puisse configurer ses touches une fois le jeu lancé. Concernant les contrôles nous pourrions sûrement intégrer l'utilisation de Joystick et joypad et éventuellement le retour de force afin de créer une interactivité entre le jeu et le joueur.

2.4 Principe des scénarios

2.4.1 Problématique

Dans sa version finale, le jeu devra proposer au joueur plusieurs scénarios, constituant une histoire crédible, et dont la difficulté augmente progressivement jusqu'à la fin de celui-ci. Pour cela, il faudra être capable de créer une suite de plusieurs missions, possédant chacune des objectifs, et un déroulement défini à l'avance, qui sera influencé selon les actions du joueur.

2.4.2 Principe

Le jeu fera donc appel à des fichiers de scénarios, possédant chacun :

- Les caractéristiques de départ du vaisseau joueur, c'est à dire :
 - Sa position de départ
 - Ses caractéristiques (vitesse, énergie, boucliers)
 - Son armement
- Les mêmes caractéristiques pour les autres vaisseaux, ennemis et alliés, incluant en plus les données concernant la gestion de ceux-ci par l'Intelligence Artificielle
- Les actions déclenchés à partir de moments prédéfinis, comme le temps de la partie, calculé depuis le départ du scénario, ou encore suivant un événement, la mort d'un des ennemis par exemple. Ces actions influenceront directement sur le jeu, quelques exemples :
 - Des messages radios
 - Le déplacement du joueur dans une nouvelle zone de jeu
 - Une avarie sur le vaisseau du joueur
 - L'arrivée de nouveaux vaisseaux, alliés ou ennemis, gérés par l'ordinateur
 - Le comportement des vaisseaux gérés par l'ordinateur
 - La fin du jeu, en victoire ou en défaite

Ces fichiers seront parsés au démarrage d'une partie, et les informations seront chargées soit directement pour une utilisation immédiate, soit dans une des deux listes d'actions, selon que l'évènement soit basé :

- Sur le temps écoulé depuis le départ du scénario
- Sur un événement

Il nous faut donc un fichier par scénario, contenant toutes ces informations, car ainsi nous aurons la possibilité d'utiliser très simplement les fichiers définissant chaque scénario

de la campagne de jeu. Cependant, la masse d'information et sa complexité (il y a beaucoup de positions dans l'espace à gérer) font qu'il serait trop long de créer ses fichiers à la main, nous avons donc commencé la réalisation d'un éditeur de scénario.

2.4.3 Réalisation de l'éditeur de scénario

Le travail à donc consisté à créer un éditeur afin de créer le fichier du scénario. Il est destiné à aider la création d'un scénario par notre équipe, cependant, vu que nous comptons l'inclure dans la version finale du jeu, il est indispensable de l'organiser un minimum afin qu'un utilisateur puisse lui-même créer ses propres niveaux de jeu. Il doit donc être présentable et compréhensible assez rapidement, il comprendra notamment une aide pour sa version finalisée.

Afin qu'il soit simple d'utilisation, il contient des protections afin d'éviter des erreurs si l'utilisateur rentre des valeurs non conformes au format attendu. Il possède aussi une carte en 2D (vue de dessus) qui permet de visualiser la position des vaisseaux, et offre la possibilité de récupérer les coordonnées et de placer un vaisseau ou un objet en cliquant directement à l'endroit souhaité. L'échelle de représentation est ajustable, on peut se déplacer sur la carte, changer les couleurs de chaque objet ou vaisseaux afin de les identifier plus rapidement.

Une fois que l'utilisateur a fini ses modifications, il peut sauvegarder le fichier qui sera directement lisible par le jeu. Il peut aussi bien sûr charger un fichier dans le but de le modifier, un parseur a été inclus dans ce but, dans lequel on a également placé une détection des erreurs.

L'ajout des nouvelles commandes dans l'éditeur interviendra au fur et à mesure de l'implémentation de celles-ci dans le moteur du jeu.

3 Avance du projet

Tâches prévues dans le cahier des charges :

Recherche et documentations diverses

Nous avons recherché et utilisé toutes les ressources nécessaires à notre entreprise, et maîtrisant déjà le langage Pascal / Delphi la documentation fût surtout nécessaire pour OpenGL, Direct3D, ainsi que DirectInput. Les ressources utilisées furent majoritairement situées sur Internet, cependant nous avons consulté quelques livres (les références des livres ainsi que les ressources tirées d'Internet sont situées dans la partie Ressources).

Ébauche du moteur graphique

Le moteur graphique est plus avancé que prévu, il nous permet déjà de charger un objet en 3 dimensions, de le faire bouger, de le texturer sommairement, ainsi que d'afficher du texte. Nous avons largement accompli notre objectif pour la première soutenance surtout vu que le moteur à été commencé en OpenGL, puis totalement recommencé en DirectX.

Gestion du clavier

Le clavier est totalement géré, avec DirectInput, et, chose dont nous n'étions pas certains d'accomplir pour la première soutenance, il gère un buffer. De plus, il permet de modifier les touches selon les habitudes du joueur.

Ébauche de l'éditeur d'événements

En ce concerne l'éditeur d'événements, nous ne nous sommes pas arrêtés à l'ébauche de l'éditeur, mais avons continués et sommes arrivés à un éditeur totalement opérationnel, qui est prêt à évoluer en même temps que le jeu lui-même se développera.

Tâches réalisées non prévues dans le cahier des charges :

Ébauche du loader 3D Studio Max

Nous avons déjà une partie du loader 3D, ce qui n'était pas prévu pour cette soutenance.

Affichage du texte

L'affichage du texte à été réalisé, selon une méthode plus complexe que prévue, ce qui nous donnera plus de possibilités par la suite

4 Ressources

Différents sites Internet visités :

- *<http://www.directxfaq.com/directinput.htm>*, un site assez intéressant sur DirectX
- *home.arcor.de/tierezuhause/delphi4all/directx.htm*, site en Allemand fournissant de bons exemple de DirectX
- *www.savagesoftware.com.au/DelphiGamer/downloads.php*, site intéressant fournissant plusieurs tutoriaux très pratique
- *<http://www.developpez.com>*
- *<http://www.jedi-project.org>*
- *<http://www.delphisource.com>*
- *<http://www.game-dev.net>*, site contenant divers tutoriaux sur l'utilisation des quaternions
- *L'intro DirectX* aux éditions CampusPress, livre de DirectX intéressant et assez clair
- *DirectX Delphi Game Programming*, Difficile d'accès, car en anglais, mais très clair et pratique pour une initiation à DirectX
- *L'Intro DirectX* - Campus Press - Difficile d'accès car les exemples sont en C++ , mais traite de parties très intéressantes au niveau de DirectX3D
- *Delphi 4 guide du développeur* - Campus Press - Très pratique par l'initiation à l'environnement Delphi
- *Turbo Pascal 7* - Eyrolles - Bien expliqué, pratique pour se former au langage Pascal

5 Prochaine Soutenance

Voici une description des travaux que nous comptons réaliser pour la prochaine soutenance :

Importateur d'objets 3D Studio Max

Le Loader 3DSMax devrait être terminé pour cette prochaine soutenance, avec la gestion des textures et offrira la possibilité de charger plusieurs objets qui seront gérés par le jeu.

Moteur Graphique

Nous améliorerons la gestion des caméras, proposerons l'affichage simultané de plusieurs vaisseaux.

Moteur physique avec gestion du déplacement

Le déplacement du vaisseau du joueur sera amélioré en fonction des touches et sera capable d'effectuer les calculs d'autres vaisseaux gérés par l'ordinateur.

Début des travaux sur la collision

Nous rechercherons des méthodes afin de proposer une détection des collisions qui ne prends pas trop de ressources, et nous serons sûrement capables d'implémenter un début de détection des collisions.

Site Internet

Nous présenterons un début de réalisation d'un site Internet présentant notre projet et son groupe, et proposer le téléchargement du cahier des charge, des rapports ainsi que du projet (une fois que celui-ci sera réalisé).

6 Conclusion

Ainsi pour cette soutenance chacun de nous a eut sa partie sur laquelle il a pu travailler et remplir sa part du contrat en terminant à temps le travail prévu. Nous sommes donc à jour du point de vue du cahier des charges et nous sommes même en avance sur ce que nous avions prévu. Cette avance pourra nous donnera une marge de manoeuvre plus grande lors des prochaines soutenances.

7 Annexes

Test de moteur réalisé en OpenGL.

Moteur réalisé en DirectX avec affichage du texte.

Capture d'écran de l'éditeur de scénario.

Logiciel créé pour le test de DirectInput.