Projet de synthèse d'images

Imac – 2ème année

Dungeon Master

L'objectif de ce projet est la création d'un jeu de type « dungeon master ». Ce projet se réalise par monome ou binôme.

Le but de ce projet est de réaliser un jeu de rôle à la mode « Dungeon Master ». Ce type de jeu propose un déplacement à la première personne dans un espace « discret » (le pas de déplacement est toujours le même et les rotations se font strictement à 90°). L'aventurier se déplacera dans un donjon, y trouvera des objets et des monstres.

A. Données du donjon

Le donjon sera constitué d'un niveau qui représente une sorte de labyrinthe contenant des trésors et des monstres. Il est entièrement décrit par un fichier texte qui est décrit ci-dessous.

Le niveau en tant que tel sera chargé à partir d'une image ppm. Vous pourrez exploiter le code couleur suivant :

- noir : mur
- blanc : couloir, pièce vide
- marron (#AA7722) : porte
- bleu (#0000FF) : eau
- rouge : « entrée » (escalier montant si plusieurs niveaux)
- vert : « sortie » (escalier descendant si plusieurs niveaux)

Le fichier image contient donc des pixels et, à la couleur de ces pixels, correspond un type de case. L'origine de la carte est en bas à gauche. Attention le cas des portes (leur orientation) pose problème et vous devrez trouver une solution pour les orienter correctement.

De plus, le donjon sera agrémenté des éléments suivants :

- Des « trésors » permettant de redonner de la vie ou de l'argent (voir section sur les personnages) ou encore peuvent être des armes.
- Des « monstres » qui attaqueront les personnages « à vue » (voir section sur les monstres)

La définition de ces éléments sera indiquée dans un fichier texte dont un exemple est donné ci-dessous.

Ce fichier descriptif commence par une ligne de commentaire qui commence elle même par le signe « # ». La ligne suivante est une chaîne de caractère indiquant la position du fichier ppm permettant de charger le niveau.

Puis le fichier contient, sur une ligne, le nombre de trésors. Puis, sur chaque ligne suivante, est indiquée la description de chaque trésor. Cette description comprend : son identifiant (un entier non signé), sa position initiale sous la forme de 2 entiers non signés indiquant la position (i,j) de la case dans la carte, son nom, et, éventuellement, son type et ses caractéristiques, son modèle 3D. Vous êtes libres de modifier (et cela sera sûrement nécessaire si vous souhaitez améliorer le projet) ces spécifications. Chaque « champ » est séparé par le caractère « : ». Par exemple, si vous décidez de mettre plusieurs niveaux, vous devrez rajouter le niveau dans le descriptif de l'objet.

Ensuite, le fichier comprend, sur une ligne, le nombre de monstre puis sur chaque ligne suivante la description de chaque monstre. La description d'un monstre comprend : son identifiant (un entier non signé), sa position initiale (comme pour les trésors), son nom et ses caractéristiques (type, attaque, défense, points de vie, le fichier du modèle 3D...).

```
#Donjon naheulbeuk
naheulbeuk.ppm
4
1:3:5:tune:1:120:argent.obj
2:14:43:tune2:1:240:argent.obj
3:16:24:fiole:2:10:fiole.obj
4:35:14:gourdin:3:3:15:arme1.obj
3
1:2:5:groark:1:15:5:40:monster1.obj
2:14:42:bzzz:2:30:0:20:monster2.obj
3:18:28:bzz2:2:30:0:20:monster2.obj
```

Un exemple de fichier de donjon

Ce format de fichier est « modifiable ». En effet, vous pouvez tout à fait utiliser d'autres formats de fichier comme le XML ou le JSON tant qu'il contient les mêmes données. **Par contre, le chargement du niveau au format PPM est obligatoire**.

B. Le personnage et sa mission

Vous devrez définir un but du jeu (sortir sain et sauf du niveau, amasser X argent, tuer X monstres...). Une fois celui-ci défini vous devrez définir des règles du jeu **simples** gérant les combats et les caractéristiques des monstres et du personnages. On peut imaginer par exemple des points de vie, un facteur d'attaque et un facteur de défense (armure) ainsi que l'argent possédé voir l'équipement possédé.

Une fois les caractéristiques du personnage défini, il sera simple de rajouter des objets permettant d'améliorer (ou de diminuer) ces caractéristiques. Vous avez toute latitude pour inventer les objets que vous souhaitez. Vous pouvez les regrouper en type. Dans l'exemple ci dessus, le type 1 permet de modifier l'argent, le type 2 la santé du personnage et le type 3 représente une arme...

C. Les monstres

Les monstres possèdent leur propres caractéristiques qui sont similaires à celles du personnage. Mais il sera important de gérer, dans le jeu, le comportement de ces monstres. Le comportement qu'il vous est demandé de coder consiste à faire avancer les monstres vers le héros et à les faire attaquer lorsqu'ils sont au contact. Mais les monstres n'avanceront que si ils « voient » le héros. Voir le héros se fait, dans un premier temps, en ligne droite. Vous pouvez également inventer une distance de vue pour vos monstres. Si ils ne « voient » rien, les monstres restent immobiles.

Les monstres, lorsqu'ils sont actifs, se déplacent ou frappent à intervalles de temps réguliers. Vous devrez donc gérer correctement, dans le temps, leurs actions.

C. Représentation graphiques du décor, des monstres et des objets

Les murs, le sol et le plafond, seront habillés, suivant leur type, par des textures prédéfinies. Le modèle d'illumination utilisée sera un classique modèle de Lambert (simple diffusion). Une seule lumière sera définie au niveau de l'utilisateur et sera dotée d'atténuation quadratique. Les portes sont un élément particulier. Elles pourront être représentées par un simple quad texturé. Il n'y a pas de gestion des ombres.

Les objets et les monstres seront dessinés soit sous la forme d'un simple quad contenant une texture, soit par un modèle 3D de votre choix. Ces textures ou ces modèles seront soit prédéfinies (suivant le type de trésor ou de monstre), soit indiqués directement dans le fichier de description du donjon.

D. Travail à réaliser

Vous devrez réaliser une application en C/C++ utilisant de l'OpenGL 3.3 ou plus ! Vous pouvez exploiter les bibliothèques vues en TD (SDL, GLEW, glimac…) ainsi que d'autres bibliothèques, par exemple de chargement de modèle 3D (ASSIMP).

Après avoir défini formellement le type de jeu et ses règles, vous commencerez vos développements par réfléchir à la structuration de votre programme et aux structures de données. Vous définirez formellement les spécifications du fichier d'entrée.

Vous devrez ensuite réaliser les spécifications suivantes :

1. Chargement de la carte : détermination du point d'entrée et de sortie.
2. Chargement des trésors et des monstres ainsi que leurs caractéristiques
3. Affichage de la carte (des couloirs du donjon)
4. Navigation simple dans le donjon
5. Interaction avec les trésors : ramassage de trésors et modifications des caractéristiques du personnage
6. Intégration et visualisation des monstres dans le donjon
7. Comportement des monstres : gestion de la « visibilité » des monstres vis à vis du héros et « animation » des monstres (déplacement vers le héros)
8. Interaction avec les monstres : combat...

. ***Chargement de la carte & Affichage de la carte***

Voir annexe 1

. ***Chargement des trésors et des monstres :***

Cette partie peut être ardue. Tâchez de bien respecter les spécifications.

. ***Navigation simple dans le donjon :***

Prendre en compte les cases « spéciales » : portes, eau …

. ***Interaction avec les trésors :***

On pourra se contenter, au clic de la souris sur la case devant soi, de vérifier la présence de trésor et, si elle est avérée, ramasser ce trésor. Cette partie implique la représentation des trésors comme vu ci-dessus. A priori aucune transparence n'est prévue mais elle peut être rajoutée (voir section « Pour aller plus loin »).

. ***Intégration et visualisation des monstres***

Comme pour les trésors, cette partie consiste à positionner et visualiser les monstres dans la scène. Vous devrez tenir à jour la position de chacun des monstres et les afficher lorsqu'ils sont visibles. A priori aucune transparence n'est prévue mais elle peut être rajoutée (voir section « Pour aller plus loin »).

. ***Comportement des monstres :***

Cette partie consiste à vérifier si le monstre voit le héros (en ligne droite) et, ensuite, faire en sorte que celui se dirige, à intervalle de temps régulier, vers le héros (tant que celui-ci est visible).

. ***Interaction avec les monstres :***

Cette partie vise à implémenter les actions du héros sur les monstres et vice versa. Les monstres frappent à intervalle de temps régulier (le même que pour le déplacement). Le héros, quant à lui, peut frapper aussi souvent qu'il le souhaite (à la souris en cliquant sur la case cible contenant le monstre).

E. Pour aller plus loin

Voici une liste non exhaustive des améliorations que vous pourrez apporter à votre projet et qui vous permettront d'avoir une bonne note.

. ***Améliorer l'affichage des trésors et des monstres :***

Le problème avec la technique du quad texturé c'est que si l'on ne gère pas la transparence, c'est moche. Il faudrait donc gérer la transparence. Pour ce faire, il faut pouvoir charger des textures contenant de la transparence. Les images avec transparence sont des images à 4 canal de couleur RVBA. Le A est la transparence. Lorsque cette « couleur » vaut 1 alors le pixel est opaque. Si il vaut 0, il est transparent.

Pour charger des images RVBA, le plus simple est de charger des images PPM et d'associer une couleur (par exemple le violet #FF00FF) à la transparence pure (=0). Vous devrez donc vous même, à partir du fichier PPM, reconstruire l'image transparente RVBA (taille de l'image 4*w*h).

Ensuite vous devrez indiquer au programme que vous utiliser la transparence avec `glutInitDisplayMode(GLUT_RGBA|GLUT_DEPTH|GLUT_DOUBLE)` puis activer la gestion de la transparence avec la fonction `glEnable(GL_BLEND)` et appeler la fonction `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`. A tout moment vous pouvez désactiver la gestion de la transparence avec `glDisable(GL_BLEND)`. Pour gérer la transparence, il faut que les couleurs et les caractéristiques des matériaux aient une composante alpha ainsi que les textures.

. ***Améliorer le comportement des monstres***

Plutôt que de « voir » le héros en ligne droite, les monstres pourraient plutôt le « sentir » à une distance déterminée de cases. Dans ce cas, il faudra que les monstres se dirigent le plus rapidement possible vers le héros (si c'est possible). Si vous arrivez à prendre en compte les murs c'est encore mieux...

. ***Plusieurs niveaux***

Votre donjon pourrait comprendre plusieurs étages. Dans ce cas, il faut indiquer pour tous les trésors et tous les monstres leur étage. De plus, au début du fichier, il faudra indiquer le nombre de niveau et lister l'ensemble des images ppm représentant ces niveaux (dans l'ordre). A tout moment, un seul niveau est chargé en mémoire.

. ***Améliorer les interactions***

On peut positionner les trésors dans leur case. On peut créer des « interactions » entre le héros et les portes (ouvrir, fermer, casser). On peut donner des caractéristiques de vitesse différentes pour les monstres (pas le même intervalle de temps). On peut rajouter des projectiles (pour le héros ou pour les monstres). On pourrait aussi rajouter un inventaire pour le héros.

F. Ressources

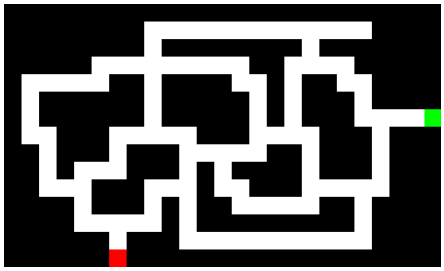
Vous pouvez trouver des ressources graphiques sur le site :
http://greatstone.free.fr/dm/ (partie games > dungeon master)

Plus d'information sur « dungeon master » sur wikipedia :
http://fr.wikipedia.org/wiki/Dungeon_Master
L'encyclopédie complète du jeu sur :
http://dmweb.free.fr/

G. Annexes : Gestion du labyrinthe

Chargement du labyrinthe

Voici un exemple de labyrinthe :



Votre labyrinthe est créé grâce à une image au format PPM. Ce format, très simple, est facile à lire et donc à manipuler. Attention la lecture de l'image se fait à l'envers (le haut en premier), il faut donc éventuellement « retourner » l'image.

Point d'entrée du labyrinthe

L'image représentant le labyrinthe contient un (et un seul) pixel rouge. C'est là que commence votre personnage. Cette entrée (ce pixel rouge) ne peut pas être dans un coin de l'image et est forcément sur un bord de l'image.
Déterminez également la première direction du regard. Le principe est simple : c'est la direction opposé au bord de l'image. La direction est exprimée, suivant les directions x et y en -1, 0 ou +1.

Dessin du labyrinthe

Pour dessiner le labyrinthe, nous allons caractériser chaque case suivant les murs présent sur ses bords est, nord, ouest et sud. Puis nous réaliserons une fonction générique permettant de dessiner une case. Enfin nous pourrons visualiser les cases directement visibles de la caméra.

Le principe est donc de définir une fonction qui dessine une case, quelque soit sont « type ». Puis de positionner la caméra sur le point d'entrée calculé précédemment et la faire la regarder « dans la bonne direction ». Puis dessinez la partie du labyrinthe qui fait face à la caméra, ainsi que les cases adjacentes à celles-ci (qui ne soient pas des murs).

Note que l'on peut aussi décider de dessiner l'intégralité du labyrinthe, c'est tout à fait permi.

Navigation

Cette partie est très simple puisque que l'on ne peut se déplacer que d'une case à une autre en 6 connectivité. Les rotations sont forcément de 90°.