Projet de Programmation Objet

Master QDCS, UE: Programmation Orientée Objet
Premier Semestre 2022-2023
Projet : Patrick Amar

Projet de Programmation Objet

Travailler en binôme :

Pour ce projet vous allez travailler avec une autre personne. Vous devez bien sûr travailler ensemble : concevoir le programme ensemble, le taper et le mettre au point ensemble sur l'ordinateur. Il n'est pas question que certains fassent le travail, et que d'autres y ajoutent leur nom ! Vous ne devrez rendre qu'un ensemble de rapports en indiquant vos noms.

Comment rendre les rapports :

Il faudra rendre un rapport et un programme : c'est-à-dire plusieurs fichiers que vous regrouperez dans une archive ZIP ou RAR ou TGZ; Vous devrez me l'envoyer par mail à l'adresse suivante : `pa@lri.fr`
N'envoyez **qu'un et un seul** mail par binôme.

Les buts de ce projet :

- Vous apprendre à écrire un programme à partir de spécifications informelles (énoncé en français).
- Vous familiariser avec le fait que vos programmes seront *réutilisés* par d'autres plus tard; donc à les écrire lisiblement en les **commentant** judicieusement.
- Vous entraîner à **encapsuler** vos données et méthodes pour obtenir des programmes robustes.
- Vous montrer l'intérêt de séparer la partie **traitement** d'un programme de sa partie **interface utilisateur**.

Ce qui vous sera fourni

On vous fournira diverses classes et fonctions C++ permettant l'affichage en 3D avec OpenGL du labyrinthe et des diverses textures.

Que devez vous faire dans ce projet ?

On se propose de réaliser le jeu de type *Kill them all* en beaucoup plus simple, avec un affichage 3D lui aussi simplifié.

La machine anime des personnages de type *gardien* dont le but est de protéger un trésor caché dans une pièce du labyrinthe, que vous, le *chasseur*, essayez de prendre.

Le chasseur se déplace dans le labyrinthe dans une direction qui est indiquée par la souris et/ou le clavier, c'est le joueur humain qui contrôle complètement le chasseur. Par contre, les gardiens sont des incarnations de la machine : ils sont animés par une sorte d'instinct artificiel qui les poussent a *attaquer* le chasseur.

Chaque personnage possède un capital initial de survie qui est décrémenté à chaque fois qu'il subit une

attaque de l'un de ses adversaires. Ce capital est incrémenté quand le personnage reste un certain temps sans subir de blessure (ou tout autre critère comme par exemple trouver des réserves de santé sous forme de caisses de survie). Le capital initial de chaque type de personnage (gardiens, chasseur) et le temps de récupération devront être des paramètres du programme.

Les gardiens ont deux **états** de fonctionnement :

- le mode **patrouille** où ils se déplacent de façon aléatoire : ils vont en ligne droite jusqu'à un obstacle, où ils choisissent une nouvelle direction aléatoirement.
- le mode **attaque** où ils attaquent le chasseur à distance en tirant dessus avec une arme qui tire en ligne droite, et dont le projectile est arrêté par un obstacle quelconque (mur, objet, etc.).
  Pour laisser une chance au chasseur, chaque gardien a un probabilité de manquer sa cible; cette probabilité dépend de l'état de santé du gardien : moins il a de points de vie, plus il tire mal. Le coefficient qui relie l'état de santé d'un gardien à sa capacité à tirer précisément est aussi un paramètre du programme.

Les critère qui décide de l'état dans lequel est un gardien est simple : un gardien qui **voit** le chasseur passe en mode attaque, sinon il passe en mode patrouille. Un gardien *voit* le chasseur quand aucun obstacle se trouve entre ce gardien et le chasseur.

Réciproquement, le chasseur peut attaquer les gardiens, il partage avec les gardiens la caractéristique d'avoir une précision de tir variable avec sont état de santé. Le coefficient n'est pas le même que pour un gardien car le tir du chasseur est dirigé par un humain (et est donc bien moins précis que ce que peut faire la machine).

Le labyrinthe

Le constructeur de la classe Labyrinthe servira à initialiser un labyrinthe selon la représentation interne du programme de jeu à partir d'une représentation textuelle du labyrinthe (à l'aide des caractères `+ - |`).

Conceptuellement le labyrinthe est en 2D, c'est l'affichage graphique qui va le rendre en 3D, vos personnages évoluent dans un plan où les murs des pièces (ou des couloirs qui ne sont que des pièces particulières) sont des lignes droites horizontales ou verticales. Le labyrinthe est *fermé*, et possède deux pièces particulières :

- celle où le chasseur est placé initialement,
- celle où se trouve le trésor.

Pour *meubler* le labyrinthe, on pourra y placer des caisses qui empêchent le passage. Des affiches (images gif ou jpg) pourront aussi être montrées sur les murs.
On repérera les positions (initiales) du chasseur, des gardiens, du trésor et des caisses dans le labyrinthe à l'aide de lettres spécifiques:

- 'C' pour le chasseur
- 'G' pour chaque gardien
- 'T' pour le trésor
- 'X' pour chaque caisse

Un affiche sera repérée par la lettre (minuscule) la représentant qui sera mise à la place du '-' ou du '|' correspondant à un pan de mur (voir le labyrinthe exemple).

Structure du programme

Pour que votre code soit lisible vous mettrez un gros commentaire au début de chaque fichier indiquant à quoi servent les classes qui s'y trouvent, comment elles s'articulent pour constituer le module et à quoi sert

ce module. Vous mettrez aussi un petit commentaire en tête de chaque méthode indiquant ce qu'elle fait, la signification de chacun de ses arguments, et celle de la valeur de retour. Enfin, mettez un petit commentaire indiquant ce que représente chaque champ de donnée de vos classes. Veillez aussi à respecter les conventions de nommage qui vous ont été indiquées dans le cours.

C'est à vous de concevoir l'architecture de votre programme de façon à ce qu'il ait toutes les qualités exigées d'un bon logiciel : simplicité, concision, clarté, etc.

Entre autres, commencez par identifier les *objets* impliqués dans le programme, indiquez comment ces objets s'articulent entre eux pour faire ce qu'on vous demande. Parallèlement, commencez à énumérer les caractéristiques de ces objets : attributs puis méthodes, pour identifier leur *classe*. Ensuite, regardez les relations entre ces classes, peut être partagent-elles des caractéristiques ? Vous pourrez implémenter ces relations avec la dérivation ou l'utilisation.

Classes dont devront dériver les vôtres

Pour pouvoir profiter de l'affichage 3D avec OpenGL, vos classes correspondant au labyrinthe et aux gardiens/chasseur devront dériver des modèles téléchargeables suivants:

- la classe `Environnement` qui définit aussi des structures pour les murs, les caisses et éventuellement des affiches.
- la classe `Mover` qui définit les fonctionnalités minimales des gardiens/chasseur qui seront utilisées par l'interface graphique.
- L'archive labh-proto-fltk.tgz qui contient tous les fichiers d'entête (`Chasseur.h Environnement.h Gardien.h Mover.h FireBall.h`) les fichiers d'exemple (`Labyrinthe.h Labyrinthe.cpp Makefile`). C'est le fichier objet à linker avec votre code pour obtenir un exécutable. La fonction `main` est contenue dans ce fichier donc ne la définissez pas vous même!
Cette archive contient aussi toutes les textures (fichiers .jpg) et le modèle de gardien (`garde.md2`).

Bien entendu, il **ne faut pas modifier** ne serait-ce que d'un caractère le contenu de ces fichiers d'entête... Sauf bien sur `Labyrinthe.h` et `Labyrinthe.cc` de vous devez récrire : votre classe `Labyrinthe` (dérivée de `Environnement`) doit posséder un constructeur qui prend un argument de type `char*` qui est le nom du fichier définissant le labyrinthe.

La constante `Environnement::scale` est un facteur d'échelle permettant de convertir des coordonnées flottantes de personnages en coordonnées entières dans la carte. Cette conversion se passe de la façon suivante: (par exemple)

```
Mover*     m;

int i = (int) (m -> _x / Environnement::scale);
int j = (int) (m -> _y / Environnement::scale);
```

Rien de tel qu'un exemple pour comprendre comment utiliser du code que vous n'avez pas écrit vous même. Téléchargez l'archive `LabyrintheProto.tgz` (voir plus haut):

- `Labyrinthe.h` et `Labyrinthe.cpp` qui définissent un labyrinthe trivial.
- `Chasseur` et `Gardien` qui sont deux exemples quasi vides de dérivés de la classe `Mover`
- `Makefile` qui vous montre les bibliothèques à utiliser pour l'édition de liens de votre application.

La classe `Mover` (et donc par conséquence ses dérivées `Gardien` et `Chasseur`) permet l'affichage d'un projectile (boule de feu) qui nécessite l'inclusion du fichier `FireBall.h` et une méthode virtuelle :

```
bool process_fireball (float dx, float dy)
```

dont un exemple est donné dans `Chasseur.h`. Cette fonction est appelée automatiquement quand on a initialisé un tir (voir la méthode `fire` du chasseur). Elle retourne vrai si le déplacement est accepté et faux

s'il y a une collision. En cas de collision, la boule explosera automatiquement.