PROJET C++: Jeu Vidéo RPG

Vincent Laurain

Le but du TP, est de réaliser une application console (nouveau projet console, vide sous visual C++) ainsi que le compte-rendu qui va avec. Ce travail **doit être personnel**. Etant donné qu'il y a une assez grande composante créative, **il est impossible que deux personnes se retrouvent avec le même programme à la fin**.

La notation sera faite à la volée durant toutes les séances de TP

Les codes .h et .cpp devront être déposés en ligne sur Arche

Table des matières

1 But du Projet **2**

2 Rendu approximatif final **3**

3 Héritage : Création de la classe personnage/ennemis (1 séance) **4**

4 Polymorphisme : le décor (1 séance) **4**

5 Tache 3 :Création de la trame du jeu (1-2 séance) **6**

6 BONUS Stage **6**

7 Annexe **7**

1 But du Projet

Introduction

Le but de ce TP, est de créer un jeu vidéo de type RPG (Role Playing Game ou "jeu de role"). Les jeux de rôles sont des jeux scénarisés qui se jouent au tour par tour. Ainsi, lorsqu'on attend une entrée du joueur, le jeu reste figé. Cela simplifie grandement l'affichage car ne nécessite pas de précision graphique ou de chronométrage des actions.

Normalement, un jeu vidéo utilise des bibliothèques gérant :

- L'affichage graphique ;
- La saisie de touches en tache de fond ;
- l'intelligence artificielle ;
- les moteurs physiques, la lumière,...

Etant donné que vous avez 12h de TP, vous L'aurez compris, le but n'est absolument pas de maîtriser tout cela **mais bien de manipuler les classes et la conception orientée objet.**

Conseils :

Le nombre de séance pour chaque tâche est à titre indicatif. Cependant, ces TPs étant une forme de projet, si vous ne deviez pas finir une tâche dans les temps, il vous incombe de rester dans les temps (travail personnel) puisque la notation tiendra fortement compte du résultat final(par exemple, si à la fin, vous n'avez fait que la Tache 1, vous ne pourrez pas dépasser 6/20). Ainsi, si vous avancez chez vous et que vous venez avec des questions concrètes sur les points qui vous bloquent, vous serez aidé immédiatement (Pour info, *"j'ai un code qui compile pas avec 10000 erreurs"* n'est pas une question concrète). Enfin si vous deviez développer sous Mac OS ou Linux (ce qui est possible si vous souhaitez bosser chez vous), certaines fonctions ne seront pas les mêmes que celles évoquées dans ce TP. **Une conversion des fonctions Windows données ici est donnée en annexe.**

2 Rendu approximatif final

Ici vous pouvez aller voir l'enseignant pour une démonstration. Parfois une démo vaut mieux que 1000 mots. **Une vidéo est également disponible sous ARCHE**. Voici Ensuite quelques précisions.

Jeu de rôle, Quesaco ?

Dans un jeu de rôle, un personnage est un aventurier qui avance dans **un décor**. A certains moment, des ennemis peuvent faire leur apparition ce qui engage **un combat** au **tour par tour** (le joueur prend une action puis c'est au tour de l'ennemi) où le joueur a un nombre de choix limité dans ses actions (attaquer, se soigner, fuir,...). **Il dispose de certaines caractéristiques** (points de vie, points de force, vitesse,...) qui sont **utilisées pour gérer les resultats de ses actions** (degats faits à l'ennemi, chance d'esquiver un coup,...). Les règles sont simples : **Le jeu se termine si le joueur atteint la fin du scenario ou si ses points de vie tombent à 0, auquel cas il meurt.**

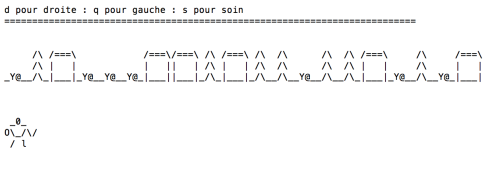
Structure graphique du jeu

En ce qui concerne l'affichage, nous utiliserons des simples fonctions comme :
— pour nettoyer un ecran : `system("cls")` (bibliotheque `stdlib.h`)
— `cout` pour afficher ;
— pour saisir une entrée (`_getch()` (bibliotheque `conio.h`) permet de prendre une entrée utilisateur et ne nécessite pas la touche entrée).

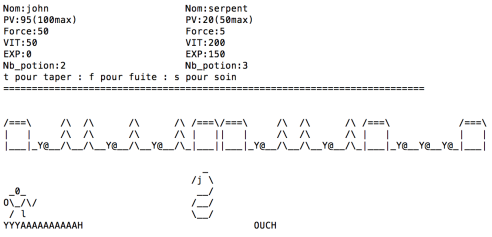
Ici il y a 3 zones d'affichage horizontales qui se superposent l'une au dessus des autres :
— Une entête (qui affiche les possibilités en mode décor, et les caractéristiques en mode combat)
— Le décor (qui constitueront la première phase du TP)
— Les personnages (qui affiche le personnage du joueur en mode "promenade", et les personnages en mode "combat")

Le jeu devrait donc avoir un rendu dans ce gout :

mode "promenade""



mode "combat"



Taches à réaliser

Voici les différentes tâches que vous aurez à réaliser qui seront détaillées par la suite :
— Séance 1 : Réalisation et test de la classe personnage ;
— Séance 2 : Réalisation du décor ;
— Séance 3-4 : Réalisation de la trame du programme et finitions ;

Bon courage, en espérant que vous y preniez plaisir !

3 Héritage : Création de la classe personnage/ennemis (1 séance)

Il y aurait beaucoup de façons possibles de créer une classe personnage. Cette classe servira, par héritage, à décrire le personnage du joueur, ainsi que les différents ennemis apparaissant. Voici une *proposition*, afin de vous aiguiller. Néanmoins, vous pouvez également vous laisser porter par votre inspiration. Attention tout de même à ne pas avoir trop d'ambitions, et assurez une fonctionnement basique avant de vous tourner vers des choses plus compliquées. Nous allons faire l'utilisation des objet `string` qui s'utilisent en incluant la bibliothèque

```
#include<string> ;
using namespace std;
```

Ici, nous allons créer plusieurs types de personnages depuis le `heros` jusqu'aux `ennemis`. Votre imagination est sans limite alors créez ce qui vous plaira. A minima, il vous faudra avoir un personnage principal ainsi que deux types d'ennemis.

Les données membres du personnage sont par exemple :

- `PV`;// *represente les points de vie du personnage, si ils tombent à 0 le personnage est mort*
- `PVmax` ; *// represente les points de vie max du personnage (PV ne peut dépasser PVmax*
- `int Force` ; *// represente la force du personnage*
- `VIT` ; *// represente la vitesse du personnage*
- `nb_potion`;// *represente le nombre de potions du personnage*
- (facultatif) `EXP;` *// represente l'expérience du personnage*
- (facultatif) `message` ; *// permet d'afficher un message pour chaque action.*
- (facultatif) `nom`;// *nom du personnage*

Les fonctions membres du personnage sont :

- Consctructeurs : vide et avec paramètres
- Getters/setters
- `Donner_coup` : //Pour le `heros`, cette fonction donne un simple coup à un autre personnage, lui enlevant des points de vie, fonction de la comparaison de leur force. Cependant pour un `ennemi`, cette fonction sera son intelligence artificielle, de sorte que chaque ennemi ait un comportement différent.
- `prendre_degat` : //Pourra être utilisée dans `Donner_coup` . Le heros prendra simplement des dégats infligés. D'autres ennemis pourront comparer leur vitesse et éventuellement esquiver certains coups.
- `soin() :` *// remonte les PV du personnage d'un certain pourcentage des PVmax. Décrémente le nombre de potions*
- `Affiche() :` Affiche le personnage sur 3 lignes.

Vous pouvez également ajouter des processus de montée en expérience, de fuite de combat....tout ce que vous souhaitez

Test de la classe

Une fois la classe réalisée, vous pouvez donc tester le fonctionnement de cette classe, **c'est à dire tester toutes ses fonctions**. Vous pourrez par exemple, dans le main, créer 3 personnages et les faire combattre ensemble et afficher les différentes données des personnages. Etant donné que vous faites des "clear screen", une fonction très utile sera `Sleep(t) (bibliothèque Windosw.h)` où `t` est un temps en millisecondes : Cela permettra de figer le programme pendant un temps donné.

Conseils

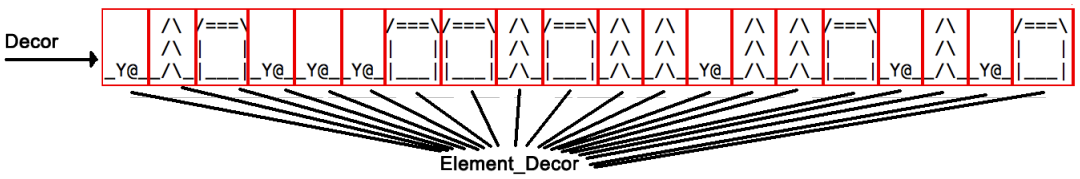
4 Polymorphisme : le décor (1 séance)

A la fin de cette séance ce stade vous êtes capables d'afficher un décor, et de le faire bouger.

Comme vous pouvez le voir sur les captures d'écran ci-dessus, un décor, comporte différents `Elements_de_decor` (j'ai fait par exemple une maison, arbre et escargot dans la figure). Chaque élément de décor est capable d'afficher chacune de ses lignes séparément :

Affiche ligne 1 `/===\`
Affiche ligne 2 `|   |`
Affiche ligne 3 `|___|`

En ce sens, un décor peut être vu comme une collection hétéroclite d'`Elements_de_decor` (utilisez le polymorphisme) et chaque `Elements_de_decor` (`maison`, `arbre` et `escargot` dans la figure) sont des classes héritées de la même classe mère.



Tâche à réaliser

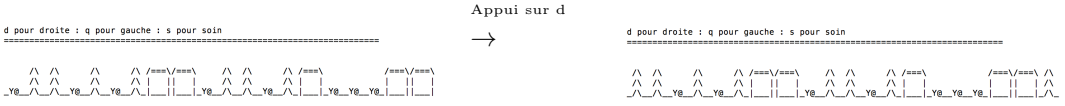
- Créer une classe `Elements_de_decor` ;
- Créer des classes `Maison, Arbre, Escargot` ;
- Dans le main :
  - Créer une variable `Decor` qui sera un tableau de pointeurs de `Elements_de_decor` de taille `M` (ils pourront être initialisés aléatoirement `rand()%int i` (bibliothèque "stdlib.h")) ;
  - En boucle, `i` s'incrémente en appuyant sur "d" (droite) et se décrémente en appuyant sur "q" (gauche) , ce qui affichera `N` "Elements de décors" entre l'élément`i` et l'élément `N+i` (choisissez `N` en fonction de la taille de la console) ;

Conseils :

La seule fonction membre des `Elements_de_decor` est `Affiche` (il n'y a pas de données membre). L'affichage doit se faire **ligne par ligne**. Ainsi on doit pouvoir afficher chaque ligne séparément (le numéro de ligne est donc un paramètre de la fonction `Affiche`).

Les questions à se poser :

- Y a t'il des classes abstraites ?
- Quelles fonctions seront virtuelles ?
- Dans l'exemple on affiche les éléments de décors sur 3 lignes. Par exemple pour une maison :
  - l'affichage de la première ligne s'écrit `cout<<"/==\\";` (attention le backslash seul se fait avec un double backslash en C)
  - l'affichage de la deuxième ligne s'écrit `cout<<"| |";`
  - l'affichage de la troisième ligne s'écrit `cout<<"|__|";`
- Dans le "main", faites attention si `i` devient inférieur à 0 et si `N+i` devient supérieur à `M` !



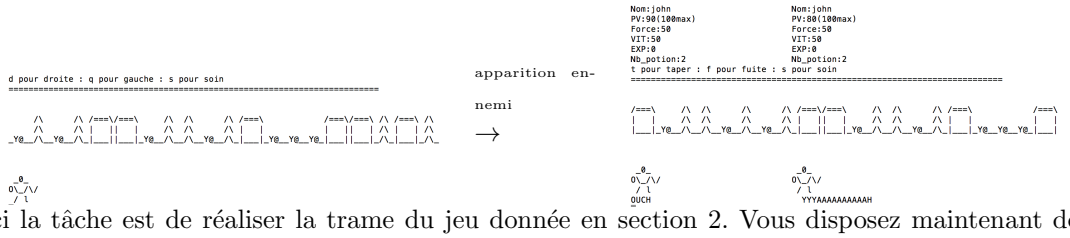
5 Tache 3 :Création de la trame du jeu (1-2 séance)

A la fin de ces séance, le projet doit être terminé. On doit pouvoir voir apparaître différents types d'ennemis, avec différentes caractéristiques. Vous devez être capable d'afficher un écran type avec un personnage et un décor qui se déplace, et un écran type ou l'on voit deux personnages et leurs caractéristiques lorsqu'ils combattent. Le personnage avance dans le décor, lorsqu'un personnage apparaît, il le combat, si il gagne, on revient au mode décor, etc...

Trame du programme

Voici la trame qui représente très grossièrement ce qu'on devra retrouver dans le main.

```
Executer en boucle:
    | Effacer l'ecran;
    | Afficher l'entete;
    | Afficher le décor;
    | Afficher le personnage;
    | Acquerir l'action du personnage (droite ou gauche);
    | Calculer si un monstre apparait:
    | si oui, passer en mode combat:
       | Effacer l'écran;
       | Afficher l'entete;
       | Afficher le décor;
       | Afficher les personnages;
       | prendre l'action du personnage et calculer ses effets (taper ou fuite ou soin);
       | faire agir l'adversaire;(on pourra par la suite ajouter une intelligence artificielle);
       | Jusqu'à ce que l'un des personnages meurt ;
Jusqu'à ce que le joueur arrive à la fin du niveau ou meurt.
```



Ici la tâche est de réaliser la trame du jeu donnée en section 2. Vous disposez maintenant de décor et de personnages. Il s'agit de réaliser les bonnes boucles afin de définir la structure du jeu. Ici les fonctions `Sleep(int t)` et `rand ()% int k` déjà évoquées seront très utiles. De plus `cout<<"\ t"` fait une tabulation afin d'avoir de beaux alignements.

La note dépendra de votre niveau de maîtrise

- **Niveau 1** : Faire la trame dans le main, sans utiliser de fonctions
- **Niveau 2** : Faire la trame dans le main, en utilisant des fonctions (nécessite les passages par référence)
- **Niveau 3** : Faire la trame en créant une classe `Jeu_De_Role`, (ainsi que de possibles nouvelles classes nécessaires comme `Decor` (nécessite d'avoir tout bien compris)

6 BONUS Stage

Dans le cas où vous seriez largement en avance ou complètement passionnés par ce genre de projet, il n'y a que votre imagination qui sera une limitation. Vous pouvez :

- Introduire des systèmes d'objet ou d'argent (données de personnage).

— Vous pourriez ajouter dans les décors, la possibilité d'entrer dans un élement de décor (par exemple une boutique autorisant des achats, un hopital pour le soin...)
— Vous pouvez mettre un système d'expérience en place qui vous rendra plus fort au fur et a mesure,
— Un scénario avec des écrans de présentation annonçant victoire et combat, un boss (un monstre plus fort) à la fin du tableau.

7 Annexe

Fonction Mac OS

— `Sleep(t)` devient `usleep(t)` (t en microsecondes) dans `<unistd.h>`
— `system("cls");` devient `system("clear");` dans `<stdlib.h>`
— `_getch();` n'existe pas, et force l'utilisation de `cin>>` (avec le besoin de taper entrée) ;
— `sprintf_s(char *,char *);` devient `sprintf(char *,char *);`