Projet IN4I11

Thierry Géraud

2017

version 1 — 13 nov

1 Le sujet et les données

Je dispose de plusieurs fichiers qui décrivent une carte / un plateau de 8×8 cases ; ci-dessous le fichier `data.txt` :

```
"        ",
"  o  o  ",
"        ",
"i   o   ",
" o      ",
"    o  T",
"   m o  ",
"        "
```

Pour charger une telle carte, il suffit de faire :

```
#include <string>
#include <iostream>

int main()
{
  std::string data[] = {
#include "data.txt"
  };
  // for (auto s : data) std::cout << s << std::endl;
}
```

Ici, on voit un individu (`i`), un monstre (`m`), des rochers (`o`) et un objectif (`T`). Le but du projet est de charger une telle carte et de simuler les déplacements de l'individu et du/des monstre/s. Par défaut, l'individu se déplace d'une case (en haut, à gauche, à droite, ou en bas), mais il ne peut pas sortir du plateau et il ne peut pas *a priori* marcher sur une case avec un rocher. Idem pour le monstre. L'individu a pour objectif de se rendre sur la case objectif objectif (`T`) ; le monstre a pour but de manger l'individu en se rendant sur la case de l'individu. La simulation des déplacements consiste à itérer : déplacement de l'individu, puis déplacement de chaque monstre (deux monstres ne peuvent pas se trouver sur une même case), puis on affiche la carte (sur `std::cout`), puis on boucle. L'exécution du programme s'arrête si l'individu ou un monstre a atteint son objectif. *Nota bene :* on se fout de savoir qui de l'individu ou du monstre va gagner ; le but du projet n'est **pas** de faire une IA...

Votre programme doit :

- avoir une représentation de la carte (intiale puis qui évolue au cours du temps / des itérations) ;
- avoir une classe par type d'objet (un rocher ou un individu est un objet...) possible ;
- être extensible pour prendre en compte de nouveaux types d'objets (cf. la section suivante) ;
- être extensible pour prendre en compte de nouvelles propriétés pour les objets (cf. aussi la section suivante) ;
- et bien sûr, jouer la simulation.

2 Les détails

Les objets "classiques" sont :

type	symbole	comportement
individu	`i`	déplacement d'1 case
objectif	`T`	case destination de l'individu
monstre	`m`	déplacement d'1 case pour se rapprocher de l'individu
rocher	`o`	case non accessible par l'individu et le monstre

Les extensions :

type	symbole	comportement
potion magique	`*`	si elle est présente, elle devient l'objectif premier de l'individu (et c'est une case non accessible par les monstres) ; lorsque l'individu atteint cette case, il peut alors de déplacer de 2 cases par tour
monstre pas con	`M`	il peut se déplacer aussi en diagonale (mais d'une case seulement)
thune	`$`	Si l'individu marche sur cette case, il a maintenant plein d'argent, s'en fout de son objectif, et le programme s'arrête (en fait, l'individu est parti vivre pépère dans les iles !) Cette case n'est pas un objectif ; l'individu ne se dirige pas vers elle, il peut simplement la croiser par hasard, en se déplaçant...

Gardez à l'esprit que votre programme (sa modélisation *orientée-objet* doit refléter son extensibilité... Exemple : on peut imaginer par la suite qu'un individu puisse marcher accidentellement sur une case `+` qui lui permette ensuite de marcher sur une case de rocher, etc.

3 Modalité de rendu

- projet par binôme (étudiant #1 + étudiant #2)
- rendu à envoyer par mail avant <**date de rendu pas encore fixée...**>
- pour le rendu :

  - envoyeur (champ `From`) : le mail ESIEE de l'étudiant #1
  - en copie (champ `CC`) : le mail ESIEE de l'étudiant #2
  - destinataire (champ `To`) à `thierry.geraud@esiee.fr`
  - sujet du mail `[IN4I11] rendu`
  - en attachement, un fichier archive de votre travail (Cf. ci-dessous)
- le fichier archive :
  - dans la suite, `prenom` et `nom` sont à remplacer par ceux de l'envoyeur
  - au format *zip*
  - le nom du fichier est `prenom_nom.zip`
  - l'archive s'extrait dans un répertoire nommé `prenom_nom`
  - tous les fichiers de l'archive doivent être dans ce répertoire...
  - ...et pas dans un sous-répertoire
  - les seuls fichiers présents doivent être en texte brut (pas d'exécutable, de `.o`, etc.)
- je dois pouvoir compiler en faisant :
  - `for i in *.cc; g++ -ansi -pedantic -Wall -Wextra -std=c++1y $i -c`
  - `g++ *.o`

  Cette boucle ne fonctionne pas forcément sur votre shell à vous (mais sur le mien, oui !)
  L'idée est que je dois pouvoir compiler séparemment chaque module (.hh + .cc) et les lier.