Rainbeau Warrior

Projet de Programmation C++ V2
Master 2ème année Mathématiques - Modélisation Aléatoire
2014/2015

1 Prologue

Azatoth est un démon qui habite un château labyrinthe. Ce serait une situation banale, qui ne gênerait personne, si Azatoth n'avait pas une mauvaise habitude : tous les mois environ, il sort de son château accompagné de ses fidèles serviteurs monstrueux et ravage, en pillant, les villages situés aux alentours de son château.

Personne ne semblait être capable de l'arrêter lorsqu'un beau jour arrive aux portes du château labyrinthe, un chevalier mystérieux qui est bien décidé à mettre fin aux débordements d'Azatoth.

2 Le projet

Vous allez devoir écrire un programme qui simule le chevalier dans le labyrinthe. Dans ce programme, l'utilisateur donne des ordres au chevalier — aller à droite, aller à gauche, tout droit, derrière — afin de le déplacer dans les salles du labyrinthe. Le but du chevalier est de retrouver Azatoth qui se cache dans l'une des salles.

Le plan du labyrinthe est lu dans un fichier texte dans un format spécial qui est décrit dans la suite (paragraphe Plan du labyrinthe) de cette section. Vous devrez absolument respecter ce format sous peine d'être pénalisé.

Vous devez au minimum implanter de quoi :
- modéliser le labyrinthe à partir du plan contenu dans le fichier.
- déplacer le personnage dans ce labyrinthe à travers une interface.

Vous pouvez également améliorer votre programme (voir section Améliorations plus bas) à condition que les deux points précédents soient parfaitement implantés.

L'interface entre le programme et l'utilisateur peut, par exemple, se faire à travers la ligne de commande usuelle.

```
...
Que décidez vous ?2
Vous arrivez dans une pièce.
Il y a  un chandelier à trois branches.
Il y a une épée, vous la ramassez.
Il n'y a pas de monstre dans la pièce.
Il y a :
```

```
        un passage à gauche
        une porte à droite verrouillée (vous n'avez pas la clef).
        une porte en arrière.
Vous pouvez
        1. aller en arrière
        2. aller à gauche.
        3. ouvrir le menu.
Que décidez vous ?_
```

Plan du labyrinthe Le labyrinthe est un ensemble de salles reliées entre elles. Chaque salle peut être reliée au plus à quatre autres salles. En d'autre termes chaque salle possède au plus quatre portes, une devant, une derrière, une dans le mur de droite, une dans le mur de gauche.

Le plan du labyrinthe du jeu doit être, par défaut, lu dans un fichier texte qui possède le nom `plan.pln`. Dans le fichier qui décrit le labyrinthe, les salles sont représentées par des nombres entiers positifs, les portes sont représentées par le nom de la salle où elle se trouve suivie par un point puis par une des quatre lettres `N,S,E,O` qui correspond à la place de la porte dans la salle.

- devant : N (pour Nord)
- derrière : S (pour Sud)
- droite : E (pour Est)
- gauche : O (pour Ouest)

La description du plan consiste à énumérer les liens entre les salles du labyrinthe. Chaque lien relie deux portes ensemble. La description du lien commence par un mot (dont le premier caractère n'est pas un chiffre) , qui peut servir à le désigner, suivie des deux portes liées, avec un tiret entre deux pour les séparer. Chaque liaison est terminée par un point-virgule.

Par exemple : `corridor1 1.N - 2.S;` signifie que si le chevalier est dans la salle 1 et qu'il emprunte la porte de devant alors il se retrouve dans la salle 2. Inversement si le chevalier est dans la salle 2 et qu'il emprunte la porte de derrière alors il se retrouve dans la salle 1. Le nom du lien est `corridor1`.

Dans le fichier le début du plan commence par le mot `debutPlan` la fin du plan fini par `finPlan`. Par défaut l'entrée du labyrinthe est la salle au numéro le plus petit et Azatoth se cache, par défaut, dans la salle au numéro le plus grand. Les portes qui ne figurent pas dans la description du plan n'existent pas : ce sont des murs infranchissables.

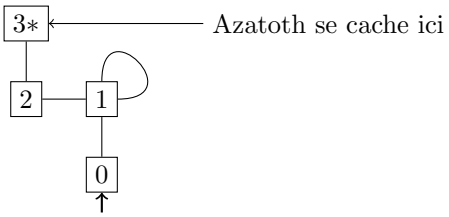
Voici un exemple de fichier plan.pln :

plan.pln

```
debutPlan
A 0.N-1.S;
A1 1.E-1.N;
A2 1.O-2.E;
B3 2.N-3.S;
```

```
finPlan
```

Ce plan décrit le labyrinthe suivant.



Entrée

Vous pouvez enrichir ce format de description en ajoutant d'autres types d'informations (monstres, objets dans des salles,...) et concevoir un programme qui les comprend (voir Amélioration). Si vous voulez ajouter des informations complémentaires sur une salle, un lien, ou une porte vous devrez l'écrire à l'extérieur du bloc `debutPlan` - `finPlan` et désigner cette salle, ce lien ou cette porte par son nom dans le bloc `debutPlan` - `finPlan`.

D'autre part même si votre programme peut gérer d'autres types de format il doit toujours rester compatible avec le format décrit ici. Des test seront effectués. Le plan lu par défaut doit s'appeler `plan.pln` et doit figurer dans le dossier courant (de l'exécution).

3 La Notation

Pour obtenir une note de Projet vous devrez :
- Fournir la totalité des fichiers sources de votre programme.
- Présenter votre programme lors d'une soutenance devant votre chargé de TP.

Les dates limites pour rendre votre projet et les dates de soutenances seront (ou sont) données sur la page d'accueil du site Didel :

`http://didel.script.univ-paris-diderot.fr/claroline/course/index.php?cid=CPP201415`

Vous avez la possibilité de faire ce projet par groupe deux ou tout seul, cependant gardez à l'esprit que deux personnes ayant travaillées ensemble n'auront pas forcément la même note de projet. Cette décision sera prise lors de la soutenance.

Soutenance Durant la soutenance, qui durera entre 10 et 30 minutes, vous effectuerez une démonstration de votre programme, présenterez brièvement les algorithmes et structures de données utilisés, et vous répondrez à d'éventuelles questions. Vous pouvez utiliser un ordinateur portable pour présenter votre programme.

Sources Le programme doit être écrit en C++. **Régle modifiée v2** : la version 1 du projet imposait que les codes sources devaient pouvoir être compilé sur les machines des salles 2004/5 cette règle n'est plus à observer désormais mais vous devez pouvoir présenter votre projet lors de la soutenance. Si vous craignez que votre projet ne fonctionne pas lors de la soutenance vous pouvez le présenter sur un ordinateur portable par exemple.

Vous devrez envoyer les codes sources de votre programme à l'adresse `sven.defelice@univ-mlv.fr`. Dans le sujet du courriel devront figurer les mots `Source M2 Projet C++` et votre nom ou vos deux noms si vous êtes à deux. Vous pouvez envoyer autant de versions que vous le voulez avant la date limite, seule la dernière sera prise en compte.

Les sources sont à envoyer dans une unique archive zip, tar ou rar avec votre nom (ou vos noms). Vous devez y joindre le fichier script ou le fichier makefile qui permet de compiler ces sources. Si vous choisissez un script il doit être écrit en bash. Ces fichiers doivent être nommés `Compile.sh` ou `makefile`.

Les fichiers sources doivent être conformes à la norme `c++11` du langage c++. Pour s'en assurer, avec le compilateur `g++` il suffit d'utiliser les deux options de compilation `-std=gnu++11` et -pedantic.

Pour les aspects graphiques (voir Amélioration) je vous suggère la bibliothèque graphique **SFML** v 1.6 qui est installée sur les ordinateurs des salles. Site de Sfml : `http://www.sfml-dev.org/index-fr.php`.

Notation La note portera sur

- **le code du programme** : sa lisibilité (commentaire, indentation), la pertinence des algorithmes et des structures de données utilisés, l'organisation du code en classe,
- **les réponses aux questions de soutenances**,
- **l'exécution du programme** : le programme ne plante pas, les améliorations implantées, l'ergonomie d'utilisation.

De façon générale, l'appréciation et la notation du programme portera davantage sur la lisibilité du code et la partie **exécution du programme** que sur les autres points.

Présenter un programme qui vérifie les deux points décrits dans la section **Le Projet** et qui s'exécute sans aucune erreur vous assurera une note au dessus de 10.

Dites-vous bien qu'un programme qui implante tous les aspects décrits dans la section amélioration mais qui fonctionne mal sur les deux points cités dans la section Projet peut vous rapporter une note inférieure à 10.

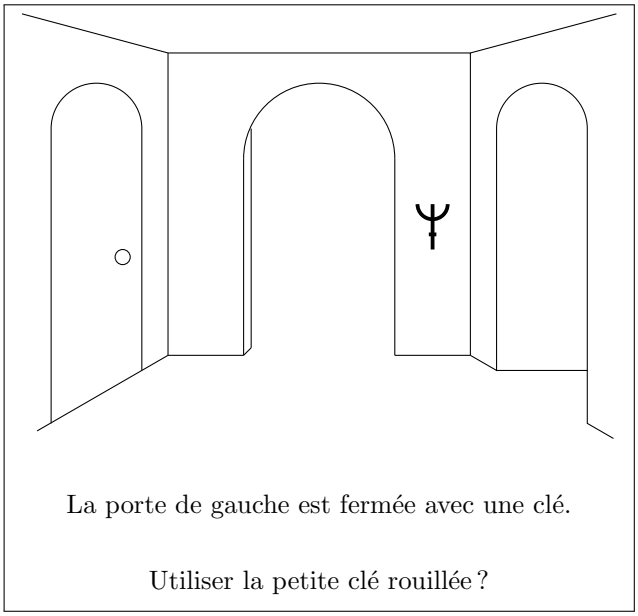
4 Amélioration

Voici une liste d'améliorations que vous pouvez ajouter à votre programme.

- Donner des ordres en appuyant sur les touches du clavier. (Utiliser une bibliothèque graphique)
- Concevoir des portes qui ne se traversent que dans un sens.

- Sauvegarder ou charger votre progression. (Lire et écrire dans un fichier)
- Spécifier le fichier lu pour le labyrinthe.
- Fermer certaines portes à clé et disperser des clés dans le labyrinthe.
- Combattre des monstres présents dans certaines salles (avec un système de combat de votre invention).
- Placer des objets (objet fixés aux salles) dans les salles permettant de les différentier (les salles).
- Dessinez les salles grâce à une bibliothèque graphique. (Voir dessin plus bas).
- Améliorer le statut d'un personnage au cours de son aventure (ex : force, argent, repérer monstres, inventaire,...).
- Génération aléatoire du labyrinthe.
- Intelligence artificiel des monstres.
- D'autres idées sorties de votre imagination (ou de votre expérience).

Affichage Vous pouvez améliorer l'affichage en utilisant des fenêtres et les outils de dessin de la bibliothèque **Sfml**
Exemple :



5 Conseil

Voici une section qui donne des conseils en vrac (la liste est susceptible de s'allonger au cours de l'année).

- Faite régulièrement des essais d'exécution de votre programme. Il vaut mieux améliorer petit à petit votre programme et faire de nombreux tests que d'écrire tout d'un coup. Les erreurs d'exécutions sont plus faciles à cerner lorsque les périodes d'écriture-test sont courtes.

- Malgré des cycles courts d'écriture-compilation-test, lorsque vous écrivez du code pensez à anticiper les améliorations que vous pourrez ajouter plus tard, quitte à écrire des fonctions vide sur le moment et à les remplir après. Les mécanismes d'héritages et d'encapsulations sont faits pour faciliter l'écriture de ces anticipations.
- Essayer d'écrire le moins de ligne de code possible. Plus le code est court plus il est facile à modifier et à corriger.
  - Si vous avez à choisir entre un code (ou un algorithme ou une structure de donnée) rapide à l'exécution mais long à écrire et un code plus court à écrire mais plus long à l'exécution, pesez bien votre choix.
    Par exemple, supposons que vous ayez besoin de faire une tâche qui sera exécutée 1 fois par seconde et que pour cette tâche vous ayez le choix entre 2 algorithmes. L'un très naïf et simple à écrire peut s'écrire en 10 lignes et s'exécute en 300 opérations (élémentaires).
    L'autre plus complexe s'écrit en 100 lignes et s'exécute en 1 opérations. Le choix (de l'avis de votre chargé de TP) penche sans hésitation pour l'utilisation de l'algorithme naïf car finalement la différence entre 1 opération par seconde ou 300 opérations par seconde ne sera pas perceptible pour un utilisateur.
  - Pour minimiser le code vous pouvez aussi utiliser le mécanisme d'héritage des classes.
- Faite attention à prévenir toutes les erreurs possibles de l'utilisateur.
- Entrer et gérer des commandes et lire un texte formaté peut se faire grâce aux méthodes de flux de la bibliothèque standard. Vous pouvez également utiliser les méthodes printf et scanf de la bibliothèque standard.
- Utilisez les outils de la bibliothèque standard du C++ qui est installée et liée par défaut (avec g++). `http://www.cplusplus.com/reference/`
- Utilisez les outils de la bibliothèque SFML (voir lien plus haut) qui est très complète il y aura sûrement un TP dessus dans l'année.
- Envoyez une version dès que possible et n'attendez pas le dernier moment, vous pourriez avoir de mauvaises surprises lors de l'envoi : Serveur en panne, Pas de Connections,...
- Certaines erreurs d'exécution peuvent être détectées par le compilateur g++ si on le lui indique. Pour activer cette option il suffit d'écrire, dans la ligne de commande de compilation, les mots `-Wall` et `-Wextra`. Exemple : `g++ fichier.cpp -Wall -Wextra`.
- Écrivez des fonctions courtes. Plus la fonction est courte plus elle est agréable et facile à lire, à modifier et corriger. Décomposez les grandes fonctions en plusieurs plus petites.