Chapitre 9

Projet

Vous contrôlez un personnage sur une carte. Récupérez le maximum de pièces d'or. Mais attention aux monstres et aux pièges.

Le but du projet est de réaliser un jeu permettant de :

1. d'initialiser une carte 20×20 dans lequel évoluera le personnage du joueur et sur laquelle seront placés des éléments comme des obstacles (rocher, arbre), des bonus (pièce d'or) et des malus (monstre, piège).
2. de permettre au joueur de déplacer un personnage sur cette carte et d'afficher la carte au fur et à mesure des déplacements.
3. de réaliser des actions en fonction de ce que le personnage rencontre sur la carte (incrémentation du nombre de pièces d'or, pertes de point de vie, etc.)

1 Carte

1.1 Initialisation de la carte

Pour représenter une carte de 20×20 nous allons utiliser une grille à 2 dimensions. Chacune des cases comportera un code représentant ce qui se trouve à cet emplacement :

— 0 : de l'herbe
— 1 : une fleur
— 2 : un obstacle : un arbre
— 3 : un obstacle : un rocher
— 4 : un objet : une clef
— 5 : un objet : une piece d'or
— 6 : un objet : un cadenas
— 7 : un piege
— 8 : un monstre

1. créer une procédure `init_carte` d'initialisation de la carte. Cette procédure prendra en argument un tableau 20×20 et initialisera chacune des cases dans laquelle vous y placerez ces différents éléments à votre convenance [1].

1.2 Affichage de la carte

Créer une procédure `affiche_carte` prenant en argument le tableau 20×20 initialisé ainsi qu'un tableau à une dimension de deux cases permettant d'afficher la carte de façon textuelle (cf. affichage ci-dessous) avec la position du personnage. La position du personnage est codée sous la forme d'un tableau de deux cases, la première représentant l'abscisse, l'autre l'ordonnée (cf. exercice 6-6) et est représentée par un `"X"` sur la carte.

1. Afin de ne pas avoir à coder chacune des 20x20 cases, on utilisera des boucles pour remplir le tableau entièrement d'herbe **puis** on y mettra les quelques autres éléments (fleurs, obstacles, objets et piège) aux endroits voulus

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 5 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 5
0 1 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 2 2 2 0 0 0 8 0 0 0 0 0 3 0 0 3
0 0 0 0 0 0 0 0 0 0 0 0 0 8 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 8 8 0 0
0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 8 3 3 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 5 5
0 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 8 3 3 3
5 0 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0 8 8 8 8
0 0 0 0 0 0 0 0 0 0 X 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 3 0 0 0 0 0 0 8 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0
0 0 3 0 0 8 8 0 0 0 8 0 0 0 0 0 0 0 0 5 0
```

2 Le personnage et son déplacement sur la carte

Il s'agit de déplacer le personnage qu'il puisse se déplacer sur la carte (dans le tableau).

2.1 Déplacement simple

Initialement, le personnage se trouve à la position (0, 0). Créer une fonction `deplace_personnage` qui prend deux arguments : le tableau 20×20 représentant la carte, et tableau 1×2 représentant les coordonnées du personnage. Cette fonction :

1. demande à l'utilisateur (à l'aide d'un `scanf`) la direction vers laquelle déplacer le personnage (8=haut, 2=bas, 6=droite, 4=gauche)
2. permet au joueur de déplacer le personnage (toujours en réutilisant ce que vous avez fait dans l'exercice 6-6). Le tableau de coordonnées sera réactualisé en fonction du déplacement (ou pas du personnage) :
   (a) en tenant compte des "bords" du tableau (déjà fait dans l'exercice 6-6)
   (b) en tenant compte des obstacles du tableau
   (c) si l'utilisateur tape la valeur 0 (au lieu des 8, 2, 6 ou 4 de déplacement), la fonction retourne -1 (arret du jeu). Sinon, elle retourne la valeur de la case sur laquelle le joueur s'est arrêté (par exemple 0 s'il s'est arrêté sur une case herbe, 5 si c'est sur une case pièce d'or, etc.)

2.2 Déplacement

En appelant depuis la fonction principale, les fonctions `deplace_personnage` et `affiche_carte`, le joueur peut déplacer son personnage et la carte avec la nouvelle position du joueur doit être réaffichée. Tant que le joueur ne demande pas à quitter, le joueur continuera à déplacer son personnage.

3 Le jeu

Un compteur de pièces d'or (initialement à 0) doit être incrémentée à chaque fois que le personnage passe sur une case contenant une pièce d'or. Un compteur de vie (initialement à 10) est décrémenté à chaque fois que le personnage passe sur une case piège. Une fois la pièce d'or prise ou le piège activé, l'objet disparait de la carte (*ie.* la case devient une simple case "herbe").

Complétez votre programme afin que :

— les compteurs de vie et de pièces d'or soient réactualisés en fonction des évènements cités ci-dessus
— ces deux compteurs soient affichés à chaque réactualisation de la carte
— le jeu se termine dans un des trois cas suivants :

1. le joueur a récupéré 10 pièces d'or. Dans ce cas, il a gagné
2. le joueur est passé à 0 points de vie. Dans ce cas, il a perdu
3. le joueur a appuyé sur "0" pour quitter volontairement le jeu.

4 Critères d'évaluation

1. le code doit être **propre** : **bien indenté** et le nom des variables explicites.
2. le code doit être correctement **commenté** avec des commentaires **utiles**.
3. les trois fonctions `init_carte`, `affiche_carte` et `deplace_personnage` doivent respecter le prototypage demandé (arguments et valeur de retour) et avoir le fonctionnement attendu.
4. l'enchainement des appels de ces fonctions afin de respecter le jeu de base tel que décrit dans le sujet doit être correct.

Bien que vous vous présenterez en binôme lors de la démonstration, vous passerez individuellement (l'un après l'autre) et donc aurez une note individuelle.

La grille de critères de notation est téléchargeable sur le site.

5 Facultatif : pour ceux qui veulent aller plus loin...

Pourvu que le sujet de base soit entièrement respecté (et à cette condition seulement).
Vous pouvez améliorer le jeu de différentes façons. Vous trouverez ci-dessous une liste non exhaustive et non exclusive des fonctionnalités supplémentaires que vous pouvez implémenter.

Notation Par soucis d'équité, seul le jeu de base sera noté. Toute fonctionnalité supplémentaire n'apportera pas de point supplémentaire. C'est juste "pour le *fun*".

5.1 Graphisme, gestion d'évènement et bruitage

La bibliothèque SDL (Simple DirectMedia Layer) est une bibliothèque très utilisée dans le monde de la création d'applications multimédias en deux dimensions comme les jeux vidéo, les démos graphiques, etc. Sa simplicité, sa flexibilité, sa portabilité et surtout sa licence GNU LGPL contribuent à son grand succès.

Cette bibliothèque a été installée sur les machines que vous utilisez à l'université.

Suggestions de fonctionnalités supplémentaires

1. Faire un affichage graphique de la carte, du personnage et des éléments (en sus de l'affichage textuel)
2. Gérer le déplacement au clavier, par les touches de flèches voire au curseur de souris
3. Réaliser des bruitages consécutifs à certaines actions (prise d'une pièce d'or, piège activé, fin de partie, etc.)

Attention : Pour utiliser les bibliothèques SDL avec l'extension `SDL_image` il est nécessaire d'y inclure un certain nombre d'options lors de la compilation [2]. Vous compilerez votre programme de la manière suivante :

```
gcc -o exemple_sdl_simple exemple_sdl_simple.c `sdl-config --libs --cflags`
    -L/usr/lib/i386-linux-gnu -lSDL_image -lSDL_ttf
```

Références

— Vous trouverez sur moodle, quelques fichiers basiques d'exemple.
— Vous trouverez divers tutoriaux sur le net, en particulier celui de Mathieu Nebra : `http://fr.openclassrooms.com/informatique/cours/apprenez-a-programmer-en-c` (anciennement sur site du zero).

2. ces options vous seront expliquées en L2

5.2 Chargement et sauvegarde de carte

L'utilisation de fonctions standards C (`fopen`, `fread`, `fwrite`, `fclose`) ou des primitives systèmes C (`open`, `read`, `write`, `close`), permettent dans un programme, de pouvoir créer, lire ou écrire dans un fichier.

Suggestions de fonctionnalités supplémentaires

1. Par l'utilisation de lecture de fichiers, vous pouvez réaliser le chargement d'une carte à partir d'un fichier et non plus par une initialisation "manuelle" de la carte dans votre programme. Vous pouvez ainsi créer plusieurs cartes dans des fichiers séparés et charger l'une ou l'autre des cartes à chaque partie (l'enchainement des cartes peut être fait par ordre de difficulté, de façon aléatoire, par choix du joueur, etc.)
2. Vous pouvez réaliser un mode "administrateur de jeu" dans lequel une interface graphique ou textuelle, permettra de dessiner sa carte puis de la stocker dans un fichier ou d'en modifier une existante
3. Vous pouvez permettre au joueur de sauvegarder une partie en cours en "écrivant" dans un fichier l'état de sa partie (coordonnées, nombre de points, carte) afin de la reprendre une autre fois.

Références

— man fopen, fclose, fread, fwrite, fseek
— man 2 open, close, read, write

5.3 Moteur de jeu plus "complexe"

1. vérification de la validité d'une carte (toutes les pièces d'or sont accessibles et pas enfermés dans des obstacles par exemple).
2. initialisation aléatoire d'une carte possible suivant une densité de monstre, piège, pièces d'or, obstacle, etc. définie. La validité de la carte devra être obligatoirement vérifiée dans ce cas.
3. les monstres peuvent se déplacer également à chaque "tour" de jeu. Leur comportement lors du déplacement peut être aléatoire ou plus "intelligent" par exemple en convergeant vers le joueur suivant un plus court chemin, en l'acculant vers un piège, en se placant devant la sortie ou les pièces d'or, etc.)
4. certains pièges (ou votre joueur) peuvent tirer à distance et avoir différentes caractéristiques (puissance, distance d'attaque, niveau d'énergie, de vie, etc.) et peuvent vous appartenir ou non. Vous pouvez ainsi converger vers un jeu de type "tower defense" dont le personnage est un "héro".

5.4 Jeu en réseaux

La programmation en réseau permet à deux (ou plus) programmes de communiquer. La programmation peut être fait en TCP (plus simple, plus fiable, plus lent et uniquement point à point) ou en UDP (moins simple, moins fiable, plus rapide, et diffusion possible).

Suggestions de fonctionnalités supplémentaires

1. Vous pouvez réaliser un serveur de jeu centralisant la carte et sur laquelle se déplaceront les n joueurs clients.
2. Vous pouvez réaliser un jeux client à client.

Références Vous pouvez vous inspirer des exemples et du support de cours de L3-I sur :
— `http://depinfo.u-cergy.fr/~dntt/supports/reseaux/cours-P1.ip-programme.pdf`
— `http://depinfo.u-cergy.fr/~dntt/supports/reseaux/sources`

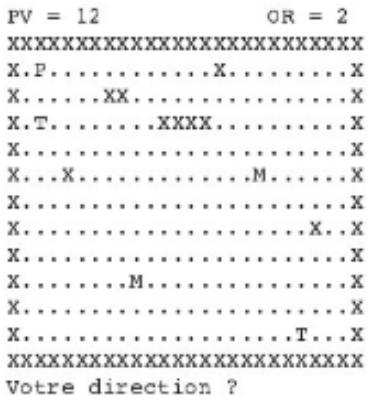
5.5 Modularité, robustesse, installation et déploiement

Suggestions de fonctionnalités supplémentaires

1. les paramètres (taille de la carte) par exemple doivent être centralisés à un seul endroit seulement (macro, fichier de configuration ou passé en argument)
2. vous pouvez regrouper certaines fonctions : utilisation de fichiers d'en-tête, création de bibliothèques statiques ou dynamiques

3. utilisation de fonctions portables et par directive conditionnelle de pré-compilation, avoir des sources portables permettant une compilation marchant aussi bien sur un système windows, linux ou mac
4. utilisation et script, makefile, autoconf pour compiler, déployer son programme
5. définition de batterie de tests pour valider la robustesse du programme (tests de couverture, de non-regression, de gestion mémoire, etc.)

Quelques captures d'écran de ce jeu réalisé les années précédentes...



6 Pour finir...

Quelques phrases à méditer :

— Il vaut mieux un programme qui fait peu de choses mais qui le fait bien qu'un programme qui essaye d'implémenter plein de fonctionnalités mais qui marche mal.
— Ce n'est pas parce qu'un programme marche qu'il est correct.
— On ne finit jamais un projet, on arrête juste de travailler dessus.