D A E M O N L A B

Rogue

Votre premier jeu vidéo

- DaemonLab -

Cette colle consiste à réaliser un petit jeu vidéo, vraiment très petit, inspiré d'un très grand jeu, ***Rogue !***

(En vérité, il est plus proche d'un petit jeu du nom de ***DungeonUp****)*

INDEX

01 – Avant-propos

Votre travail doit être rendu via le dossier **~/colle/rogue/** dans votre espace personnel.

Si vous faites erreur et que le dossier que vous utilisez pour votre rendu est différent, vous ne serez pas évalué faute d'avoir pu trouver votre travail.

Ce travail est a effectuer seul. Vous pouvez bien sur échanger avec vos camarades, néanmoins vous devez être l'auteur de votre travail. Utiliser le code d'un autre, c'est **tricher**. Et tricher annule **toutes** les médailles que vous avez reçu sur l'activité. La vérification de la triche est réalisée de la même manière que la correction : de manière **automatique**. Prenez garde si vous pensez pouvoir passer au travers.

Votre rendu doit respecter **strictement** l'ensemble des règles suivantes :

- Il ne doit contenir **aucun** fichier objet. (*.o)
- Il ne doit contenir **aucun** fichier tampon. (*~, #*#)
- Il ne doit pas contenir votre production finale (programme ou bibliothèque)

La présence d'un fichier interdit mettra immédiatement fin à votre évaluation.

Votre programme doit respecter les Tables de la Norme dans leur intégralité. Vous êtes invité à les observer depuis **l'Infosphère**. Elles sont disponibles comme ressource de cette activité.

02 – Fonctions autorisées

La bibliothèque logicielle venant avec le C est vaste et disponible. La LibLapin, que vous utilisez dans vos projets multimédia, est également vaste… Cependant nous avons fait le choix de vous interdire leurs utilisation intégrales, afin de vous amener progressivement à reprogrammer vous même ses fonctionnalités les plus utiles.

L'utilisation d'une fonction interdite est assimilée à de la triche. La triche provoque l'arrêt de l'évaluation et la perte des médailles.

Vous n'avez le droit d'utiliser aucune fonction issue de la LibC ou de la LibLapin à l'exception de celles que nous vous autoriserons explicitement.

Pouvoir utiliser une fonction ne signifie pas nécessairement que celle-ci soit utile à votre cas.

Pour cette activité, issu de la LibC, vous n'avez le droit qu'à la liste suivante :

- open
- close
- read
- srand
- cos
- atan2
- opendir
- readdir
- malloc
- stat
- getpwuid
- localtime
- write
- alloca
- atexit
- rand
- sin
- sqrt
- closedir
- mkdir
- free
- getgrgid
- strftime
- time

03 – Méthode de construction

Il peut vous être demandé d'écrire des programmes ou des fonctions.

Dans le cas des programmes, il vous sera toujours demandé de fournir un **dossier** pour l'exercice le requérant. Un **Makefile** vous sera également demandé. Le **nom du programme** de sortie vous sera précisé à chaque fois. Un Makefile incorrect, un mauvais nom de programme, et votre correction n'aura pas lieu…

Dans le cadre des fonctions, il vous ai demandé de fournir le fichier dans votre **dossier de bibliothèque personnelle**, de sorte à ce que vous puissiez utiliser toutes les fonctions que vous avez déjà réalisé jusqu'ici. Pour rappel, le dossier de votre bibliothèque doit être placé à la racine de votre espace personnel et s'appeler **libstd/.**

N'oubliez pas d'entretenir avec soin votre dossier **libstd/** de sorte à ce qu'il soit toujours propre, respecte la norme et soit en état de compiler… sans quoi elle fera obstacle à la correction.

Votre compilation devra toujours comporter les options **-W**, **-Wall** et **-Werror**.

Dans le cadre de la programmation multimédia, le système de correction établira toujours la variable d'environnement **BMALLOC** à 1. Si vous utilisez le modèle de projet, cela provoquera l'utilisation de **bunny_malloc** dans votre bibliothèque personnelle comme dans votre projet rendu.

04 – Charger, montrer, sauvegarder

Vous allez réaliser un programme. Ce programme prendra en paramètre un unique fichier carte. Ce genre de fichier, appelons le « format carte » :

```
12,6,3,next_level.map
############
#T...M.....#
#.#.##.#.#.#
#.####.#.#.#
#.M......P.#
############
```

En première ligne, vous trouverez la largeur de la carte, suivi d'une virgule suivi de la hauteur de la carte. Le troisième nombre est le nombre de vie du joueur. Le dernier paramètre de la première ligne est le prochain niveau à charger une fois que celui-ci est gagné. Ce dernier paramètre est optionnel (Il n'y aura alors pas de virgule après le troisième paramètre)

Le caractère '#' indique un mur. Le caractère 'P' indique la position de départ du joueur. Les caractères 'M' indiquent des monstres et le caractère 'T' indique le trésor.

Votre objectif est déjà de charger ce fichier. Ensuite, votre programme affichera un « prompt », c'est à dire une petite suite de symbole servant à indiquer au joueur qu'il peut taper quelque chose au clavier :

```
$> ./rogue map
ROGUE> show
############
#T...M.....#
#.#.##.#.#.#
#.####.#.#.#
#.M......X.#
############
```

La commande « show » lui servira à afficher la carte. Commencez par réaliser ce travail. Le 'X' est la position actuelle du joueur.

La commande « save » lui permettra de sauvegarder la partie, c'est à dire la carte, dans un fichier qui s'appellera comme la carte suivi de l'extension « .save », au format carte.

05 – Se déplacer, frapper, gagner

Vous allez implémenter une commande supplémentaire : la commande « **status** ». Elle affichera le nombre de point de vie du joueur, suivi d'un saut de ligne.

Vous allez maintenant ajouter quatre commandes supplémentaires : les commandes « **up** », « **down** », « **left** » et « **right** » qui serviront à déplacer le joueur sur la carte. Les commandes « **z** », « **q** », « **s** » et « **d** » auront le mème effet. Ces commandes peuvent être chaînées : « **zzzqqs** » par exemple.

- Si le joueur se déplace vers un mur, rien ne se passe.
- Si il se déplace vers un monstre, il perdra un point de vie et il n'est pas déplacé.
- Si le joueur se déplace sur le trésor, il **gagne** la partie, le programme charge le prochain niveau après avoir affiché « **Victory !** » suivi d'un saut de ligne.

Ensuite, la commande « **hit** » permettra au joueur de frapper un monstre. Si un monstre se situe à gauche, à droite, en haut ou en bas, il sera détruit. Le fait de frapper un monstre fait perdre un point de vie au joueur par monstre frappé.

Si le joueur n'a plus de point de vie, il perd la partie, le programme quitte après avoir affiché « **Defeat !** » suivi d'un saut de ligne.

05 – Powerups, monstres, pièges

Vous allez maintenant ajouter un nouveau type de « tuile » : un powerup : la tuile 'H', qui fait gagner... 3 points de vie au joueur !

De plus, désormais, la tuile 'N', également monstre, va, à chaque action de déplacement du joueur, également se déplacer : chaque monstre 'N' fera un déplacement aléatoire en haut, à gauche, en bas, à droite, **si c'est possible** d'une unique case. Les monstres n'iront jamais sur la carte trésor, ni sur un autre monstre, ni sur un mur ou un powerup.

La commande « **wait** » ou « **x** » permet de ne pas bouger ce tour ci. Utile pour attendre qu'un monstre s'approche !

Vous ajouterez également une tuile 'C', la chausse-trappe : elle n'apparaît pas quand le joueur demande à afficher la carte. Quand il marche dessus, çà écrit « **Outch !** » suivi d'un saut de ligne et lui fait perdre un point de vie. La chausse-trappe disparaît après qu'on ai marché dessus. Les monstres ne marchent pas dessus.

Le joueur peut sentir les chausses trappes à l'aide de la commande « **detect** » : si une chausse-trappe est sur une case à moins de 3 de distance dans l'une des quatre directions, « **I feel a trap !** » est affiché suivi d'un saut de ligne. Les trappes apparaissent alors lorsqu'on affiche la carte. Si aucune trappe n'est a coté, « **I don't feel any trap !** » est affiché suivi d'un saut de ligne.

06 – « Vidéo »

Bonus : Concevez une interface graphique à votre jeu. Vous devez quand même proposer l'interface texte, mais en plus, proposez une interface graphique. Le choix se fait avec une option envoyé à votre programme.