Projet 2020-2021: Bring Back Money.

Adapté du cours de l'université de Cergy.

Projet à rendre pour le 1er Février 2021 avant 20H00.
Ce TP est un travail individuel.

Adresse de contact: etienne.menager@ens-rennes.fr

Le but de ce projet est de mettre en application les concepts vus en algorithmique/ programmation cette année afin de réaliser un petit jeu appelé «Bring Back Money ».

ATTENTION: Lire attentivement l'ensemble du document avant de commencer ce travail.

1. Règle du jeu

Les règles de ce jeu sont simples: vous contrôlez un personnage sur une carte et le but est de récupérer un maximum de pièces d'or en évitant les monstres et les pièges. Le jeu est donc constitué d'éléments sont les fonctions sont:

1. L'initialisation d'une carte 20x20 (monde carré de cases carrées) dans laquelle évoluera le personnage du joueur et sur laquelle seront placés des éléments comme des obstacles, des bonus (pièce d'or) et des malus (monstres, pièges).
2. De permettre au joueur de déplacer un personnage sur cette carte et d'afficher les éléments sur la carte.
3. De réaliser des actions en fonction de ce que le personnage rencontre sur la carte (incrémentation du nombre de pièces, pertes de point de vie, etc).

Pour plus de simplicité, le jeu est dans un premier temps réalisé en tour par tour.

Les différents éléments pouvant apparaitre sur la carte sont représentés dans le tableau 1:

Élément	Représentation	Description
Joueur	J	Le joueur qu'il faut déplacer sur la carte.
Pièce d'or	O	Les pièces à collecter (collecte automatique).
Coffre	C	Des coffres qui peuvent soit contenir soit des pièces, soit des pièges.
Clé	K	Des clés pour ouvrir les coffres.
Cabane	H	Le spot de départ du joueur. Le joueur peut stocker des pièces dans la cabane sans que les monstres puissent lui voler.
Piège	P	Caché au départ, il n'apparait pas sur la carte jusqu'à ce qu'il soit découvert. Les effets sont variables: paralysie pendant un nombre donné de tour, retour à la case H, perte de point de vie.
Monstre	1, 2, 3, …	Différents types de monstre: stupide ou intelligent (se dirige vers le joueur ou non), rapide ou non (mouvement d'une case ou plus), qui prennent de la vie ou des pièces.
Obstacle	X	Un obstacle qui empêche la bonne circulation du joueur.
Herbe	G	De l'herbe qui permet de se cacher d'un monstre intelligent.

Tableau 1: *Les différents éléments pouvant exister dans le jeu.*

Un exemple de monde avec les différents objets est représenté dans la figure 1. Il faut noter que lorsque le joueur n'a plus de vie ou que le temps est écoulé la partie se termine. S'il n'y a plus de pièce, il est possible d'en ajouter de manière aléatoire.

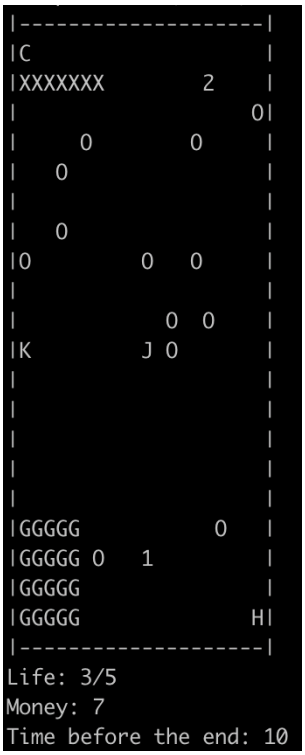

Figure 1: *Affichage graphique du jeu, exemple d'une carte 30x30.*

Le but du jeu est d'avoir le plus de pièce possible dans la cabane avant la fin du temps imparti.

2. Objectifs

L'objectif de ce projet est de vous évaluer sur les différentes connaissances et concepts suivants:

- Algorithmique et conception.
- Programmation en C.
- Structure de données.
- Gestion des entrées/sorties.
- Qualité de code (conventions, documentation, analyse statique).
- Automatisation de la compilation.
- Tests unitaires et intégration continue.

3. Cahier des charges

Dans une première partie, le travail sera guidé afin de mettre en place le monde et les différentes interactions pouvant exister dans ce monde. La deuxième partie sera de l'ordre du développement logiciel où le but sera d'utiliser les éléments de la première partie afin de réaliser un jeu complet.

ATTENTION: les variables globales sont très fortement dépréciées.

PARTIE 1

Initialisation du monde, affichage graphique

- Comment stocker l'information du monde?
- Créer une fonction permettant d'initialiser la carte. La carte doit être passée en paramètre. La position de la Cabane est aléatoire sur le monde. Différents paramètres peuvent venir moduler la création du monde: nombre de pièces, nombre et types de monstres, nombre de coffres, nombre de pièges, … La position des éléments dans le monde est aléatoire. Lorsqu'aucun objet n'est sur une case, elle est représentée par le caractère ' ' (espace). Le joueur est représenté par un tableau de taille nx1, où n>2 et représente les informations sur le joueurs (position, nombre de vie, nombre de clés, nombre de pièce sur le joueur et dans sa cabane, …). Le joueur est initialement dans sa cabane.
- Créer une fonction permettant d'afficher la carte de manière textuel, comme représenté dans la figure 1.

Déplacement du joueur

Initialement le joueur se trouve dans sa cabane. Créer une fonction permettant de déplacer le personnage. Elle prend deux arguments: la carte et le tableau du joueur. Cette fonction:

- Demande à l'utilisateur la direction vers laquelle se déplacer (« Z »: haut, « Q »: gauche, « S »: bas, « D »: droite).
- Permet de se déplacer en tenant compte de l'environnement (bord du tableau, obstacle, …).
- Retourne la valeur de la case dans lequel est le joueur.

Interaction avec l'environnement

Un compteur de pièces d'or (initialement à 0) doit être incrémenté à chaque fois que le personnage passe sur une case contenant une pièce d'or. Autrement, lorsqu'il est sur une case avec un objet, le joueur peut interagir avec les objets en tapant la touche « I ». Ceci permet d'attraper une clé, d'ouvrir un coffre, de déposer des pièces dans la cabane. La touche « I » ne compte pas comme un déplacement: si elle est utilisée, il faut tout de même permettre au joueur de se déplacer.

- Implémenter la fonction permettant d'interagir avec l'environnement. Les compteurs de clés et de pièces doivent être mis à jour au fur et à mesure du jeu. Une fois utilisés, les objets disparaissent de la carte.

Le joueur à initialement un compteur de vie plein, qui peut être modifié par les monstres et les pièges. Il existe différents types de monstres.

- 1: le monstre 1 se déplace d'une case par une case de manière aléatoire.
- 2: le monstre 2 se déplace dans un cercle de 3 case autour de lui de manière aléatoire.
- 3: le monstre 3 se déplace en direction du joueur une case par une case (**fonction récursive** de plus court chemin). Si le joueur n'est pas visible (caché dans les herbes), similaire à 1.
- 4: le monstre 4 se déplace de trois cases en direction du joueur (**fonction récursive** de plus court chemin). Si le joueur n'est pas visible (caché dans les herbes), similaire à 2.
- 5, 6, 7, 8: ces monstres sont les mêmes que les monstres précédents mais ne s'intéressent qu'à l'argent du joueur. S'il n'a plus d'argent, ils peuvent devenir violent.

Si un monstre atteint le joueur, celui ci est affecté (perte de sous ou de santé). Les monstres disparaissent alors de la case occupé par le joueur et réapparaissent sur un bord du plateau de manière aléatoire. Si le joueur n'a plus de santé la partie est finie.

- Implémenter les fonctions permettant de diriger les monstres. Penser à réutiliser le code déjà fait: les monstres partagent des comportements similaires.

Les pièges sont posés de manière aléatoire sur la carte et ne sont pas visible par le joueur. L'effet du piège est aléatoire: renvoie le joueur dans sa cabane, fait perdre des pièces ou de la santé au joueur, immobilise le joueur pour un temps donné.

- Implémenter la fonction permettant de réaliser l'interaction entre le joueur (passé en paramètre) et un piège. Il faut indiquer au joueur qu'il a rencontré un piège!

Il faut noter qu'un coffre contient soit de l'argent soit un piège, qu'il faut donc déclencher.

- Afin de faciliter le déroulement du jeu différents compteurs sont utilisés: compteur de pièces (sur le joueur et dans la cabane), compteur de vie, timer. Afficher à chaque itération ces compteurs en dessous de la carte.
- Implémenter la fonction d'arrêt qui quitte le jeu lorsque le temps est écoulé, lorsque le joueur est mort ou qu'il demande à quitter la partie (avec la touche 'Q').

PARTIE 2

Il faut maintenant intégrer ces différents éléments dans un programme global. Ce programme contient en plus les éléments suivant:

- Un menu. Il s'agit d'afficher des informations de manière graphique dans le terminal et donc d'interagir avec le jeu pour en modifier le comportement. Le menu permet de modifier les caractéristiques du jeu (difficulté en changeant le nombre de pièces/de monstres/de piège, la taille du monde, la durée des partie). Un sytème de duel est ajouté: il permet d'ajouter un joueur sur la carte, chaque joueur ayant sa maison et jouant l'un après l'autre. Le joueur qui gagne est celui qui a le plus de pièce dans sa cabane à la fin du temps réglementaire/ lorsque les deux joueurs sont morts.
- Un système de sauvegarde. L'état d'avancement de la partie doit être sauvegardé dans un système de fichiers afin de pouvoir y revenir par la suite. Si une partie est finie, un fichier est généré afin d'avoir un historique des parties jouées (score, durée depuis le début de la partie, …). Un leaderboard permet de savoir qu'elles sont les statistiques de 3 meilleurs parties.

Idée pour aller plus loin:

- Système de championnat: faire jouer plusieurs joueurs dans un championnat (poule, 8ème de finale …) comme pour un championnat de tennis. Les parties se font toujours en 1 contre 1.
- Monde ouvert: le monde n'a pas de bord et la carte évolue en fonction des déplacement du joueur.
- Brouillard: le monde est dans le brouillard et la vision du joueur est limité à 4 cases autour de lui.
- Comportement des monstres différents: les monstres peuvent être soudoyé pour éviter de perdre de la santé.
- Utilisation de la bibliothèque SDL (Simple DirectMedia Layer) pour un rendu visuel plus concret.

Toute amélioration du programme de base est pris en compte dans l'évaluation finale.

4. Délivrables

Le déliverable attendu est une archive à envoyer par mail au plus tard le 1 Février 2021 à 20H00. Cette archive devra contenir:

- Le code source de votre programme dans le répertoire src pour les fichiers *.c et include pour les fichiers *.h.
- Un fichier Makefile qui permettra de compiler automatiquement votre programme.
- Un fichier README.md qui devra présenter brièvement votre programme (comment le compiler, comment l'utiliser).

- Un rapport au format PDF: réponse au question, présentation des solutions, analyse du code et des performances, démarche de développement logiciel.
- Une vidéo (capture d'écran) du fonctionnement de votre jeu. Le code ne sera pas compilé sur la machine de l'enseignant.

5. Évaluation

L'évaluation de ce projet sera faite en fonction du code produit **et du document de présentation fourni**. Les éléments pris en compte dans cette évaluation sont: le respect du cahier des charges, la fonctionnalité et la qualité du code fourni (propreté, réutilisation de fonctions, commentaires utiles, …), la réalisation de test unitaires pour chaque fonction faites (fichier .c qui permet de tester toutes les fonction), la description complète et détaillé des choix qui ont été faits pour l'implémentation.

Les principes de programmation suivant seront également évalués:

1. Limiter la porter des données: certaines données ne doivent vivre que dans le bloc où elles interviennent.

2. Écrire des commentaires utiles: les commentaires doivent plus dire pourquoi que comment.

3. Utiliser des noms qui sont représentatifs de la fonction, de la variable, …

4. Principe KISS: « Keep It Simple Stupid ». Il faut garder les choses les plus simples possibles.

5. Principe DRY: « Don't Repeat Yourself ». Ne répéter pas du code, faite des fonctions pour réutiliser des éléments déjà implémentés.

6. Principe de responsabilité unique: chaque module/fonction ne doit se préoccuper que d'une tâche.

7. Planifier à l'avance: il faut d'abord résoudre le problème avant de commencer à coder.

Exemple de barème de l'année dernière:

1. Respect du cahier des charges (12/20)
   - Réalisation d'un affichage graphique (affichage du plateau, menu).
   - Réalisation d'un coup.
   - Différents adversaires: joueur, au moins deux IA.
   - Un des éléments suivant: sauvegarde, championnat.
   - Code en adéquation avec le travail demandé: compréhension du sujet, fonctions pertinentes pour y répondre, …
2. Rapport / présentation du travail réalisé (6/20)
   - Explications des étapes de travail
   - Explications des problèmes et des solutions apportés
3. Règle de codage (2/20)
   - Utilisation de fonctions (principe DRY, principe KISS)
   - Noms représentatifs
   - Limiter la porté des données
   - Commentaires