Méthodologie de la conception et de la programmation partie I
TP 3 : jeu

Ceci est un cours de méthodologie. Nous serons particulièrement attentif à la façon dont vous codez. Ce projet est à faire en binôme (de deux personnes, pas plus pas moins. Si vous ne trouvez pas un binôme ou si vous avez des problèmes avec lui, merci d'envoyer un mail à votre enseignant d'EI.).

Ce projet est à rendre pour le 23 mars 2023. *Vous devez le travailler en TP, en EI et en travail personnel en dehors des horaires de l'emploi du temps.*

Un gros projet

Comme Diablo IV met du temps à sortir, vous allez prendre les choses en main et faire une version alternative, bien mieux ! ou pas. Diablo IV est un *Hack 'n' slash* dans un univers fantastique, avec des monstres, des magiciens et des dragons !

Ce sera un jeu solo se jouant sur un plateau carré. Les monstres apparaitront aléatoirement sur la carte régulièrement pour détruire l'univers. Votre personnage devra les supprimer, il est le seul rempart contre le chaos...

Les règles

- Il y aura au plus un monstre par case.
- Vous pourrez être sur la même case qu'un monstre.
- Vous ne pourrez engager un combat qu'en vous déplaçant sur un monstre.
- Les monstres ont un certain nombre de points de vie (PV). Chaque attaque leur enlève un nombre aléatoire (entre 10 et 20 par exemple) de PV.
  Un monstre atteignant 0 PV meurt et disparait de la carte.
- Le joueur peut se déplacer d'une case sur la gauche, la droite, le haut ou le bas, par exemple en utilisant les touches `wasd`. Il ne peut pas sortir des limites de la carte.

Les modules

Le découpage de ce jeu a été vu en TP. Il se composera de au moins 3 modules et un fichier/module principal. Chaque module devra être indépendant des autres, comporter un fichier header, un fichier source, un fichier test et une documentation doxygen.

Vous pouvez aussi avoir un fichier header commun à tous les modules et contenant des constantes. La seule constante obligatoire est `TMAP`, la taille de la carte.

Les modules (obligatoires, vous pouvez en ajouter d'autres) seront :

0.0.1 Le joueur

Le joueur est défini par une structure `joueur_t` comportant au moins un tableau de deux entiers `pos`, la position du joueur sur la carte.

Le module devra implémenter et tester au moins les fonctions :

- `int deplacement(struct joueur_t* j)` qui demande au joueur de saisir une touche parmis `wasd` et fait le déplacement correspondant. Elle rendra `1` si le déplacement s'effectue avec succès, `0` sinon (par exemple en atteignant les bords de la carte).

— `struct joueur_t* creationJoueur()` crée un joueur. Vous aurez besoin de `malloc(sizeof(struct joueur_t))`. Vous pouvez ajouter les arguments dont vous avez besoin à cette fonction.
— `char* toStringJr(struct joueur_t* j)` renvoie une chaîne de caractères destinée à être affichée et contenant les données du joueur, pour les besoins des tests.

0.0.2 Les monstres

Ce module implémente deux structures : `monstre_t` et `listeMst_t`. `listeMst_t` a deux champs : `struct monstre_t* listeM[TMAP*TMAP]` un tableau de pointeurs vers les monstres et `int nbMst` la taille effective du tableau.

Il implémentera les fonctions :
— `struct monstreListe_t* creationListeM()` crée un tableau vide de taille `TMAP*TMAP` de monstre.
— `int ajoutMst(struct monstreListe_t* listeM, int j_pos0, int j_pos1)` Ajoute un monstre avec une position aléatoire à la liste. Le monstre naitra avec un nombre de PV aléatoire. Il ne devra pas naitre sur une case sur laquelle est présent le joueur (position donnée par `int j_pos0` et `int j_pos1`) ou un autre monstre.
— `void enleverMst(struct monstre_t* m, struct monstreListe_t* listeMst)` enlève le monstre de la liste. La liste ne doit pas avoir de trous.
— `char* toStringMst(struct monstre_t* M)` renvoie une chaîne de caractères destinée à être affichée et contenant les données du monstre, pour les besoins des tests.
— `char* toStringLstMst(struct monstreListe_t* listeM)` renvoie une chaîne de caractères destinée à être affichée et contenant les données de la liste des monstres, pour les besoins des tests.

0.0.3 L'affichage

Ce module implémentera la fonction :
— `void afficher(...);` affichera la carte avec les monstres (représentés par un M par exemple) et le joueur (par un J). On vous laisse le choix dans les paramètres de la fonction.

0.0.4 Le main

Ce module ou ce fichier (comme vous voulez) inclura (au moins) les trois autres et implémentera les fonctions :
— `int attaque(struct joueur_t* j, struct monstre_t* m);` gère une attaque du joueur sur le monstre. Cette attaque ne peut être lancée que si ces deux participants sont sur la même case. Elle retournera `1` si le monstre meurt, 0 sinon.
— `int** creationPlateau();` crée la carte (si besoin). Un tableau de tableaux d'int de taille `[TMAP][TMAP]`.
— `char** generationListeAff(struct joueur_t* j, struct monstreListe_t* listeM);` génère à partir de la liste des monstres et de la position du joueur le tableau de tableaux de char contenant les caractères à afficher. La tableau généré sera de taille `[TMAP][TMAP]`.
— `void jouer()` lance une partie. C'est la seule fonction qui devra être appelée dans le `main`.

Extension

Ce jeu est vraiment minimaliste. Pas sûr que Blizzard en veuille en l'état... Étendez-le en créant des modules `joueurs`, `monstre` ou `affichage` alternatifs.

Voici une liste non exhaustive les extensions possibles :

— Améliorer l'affichage (par exemple en faisant un plateau à une dimension mais défilant).
— Améliorer l'affichage avec SDL.
— Organiser un système de points d'action (PA) pour le joueur : chaque déplacement coût n PA, chaque attaque m PA. Vous pouvez aussi donner des coûts en PA différents en fonction des cases de la carte.
— Joueur à plusieurs joueurs. (le système de PA devient intéressant que à plusieurs joueurs)
— Modifier les règles d'attaques : attaque à distance, de zone. Avec une défense, une armure etc.
— Mettre des objets et un inventaire.
— Faire bouger les monstres aléatoirement et leur permettre d'attaquer.
— Donner un système d'experience avec des bonus et des capacités à améliorer.

Tous ces modules ne sont pas de la même difficulté. Ne soyez pas trop ambitieux et parlez en avec votre responsable de TP/EI AVANT de vous lancer dans la programmation. Si vous avez d'autre idée, discutez en AVANT avec votre responsable de TP/EI.