Rapport PG110: Projet de programmation C -Bombeirb-

Hicham Larchi
Damien Migel

mai 2019

Table des matières

1 Introduction

Une princesse est détenue prisonnière par de méchants monstres verts. Votre mission, si vous l'acceptez, est d'aller la délivrer. Pour cela, vous devrez traverser plusieurs mondes, plus effrayants les uns que les autres. Des portes vous permettront de passer de mondes en mondes. Certaines portes seront fermées à clés et nécessiterons d'avoir une clé dans votre inventaire. Vous êtes un expert en explosif et utiliserez vos bombes pour détruire les obstacles devant vous et tuer les monstres qui vous attaquerons.

2 Travail à fournir

2.1 Gestion des déplacements

Pour limiter les mouvements du player au cadre de la carte, on crée un fonction *map_is_inside*(struct map* map, int x, int y) qui retourne 1 si la position (x,y) est a l'intérieur de la map, 0 si non.
Pour limiter les mouvements au élément de décours et les caisses on a modifier la fonction *player_move_aux*() pour qu'il retourne 0, si le déplacement est impossible.
Pour déplacer, les caisses ont a crée une fonction *boxe_move*() qui verifie que la 2^{eme} la cellule dans la direction du player est vide, si c'est le cas alors le player prend la place de la caisse dans la map et cellule de la caisse est copier dans la cellule suivant.

2.2 Gestion des mondes

Le programme lis les fichiers map.txt conformément au a ce qui est demander. Le programme lis tout d'abord la longueur et la largeur de la map. Puis il lis les $longueur \times largeur$ caractères suivent du fichier.

2.3 Chargement des cartes

Le jeu charge l'ensemble des cartes un niveau dans un tableau de *map**, pour chaque *map* on initialise la *monster_list* et la *bomb_list* (voir plus loin). Pour chaque monstre dans sur la *map* un ajoute un éléments dans la *monster_list*.

2.4 Gestion du panneau d'informations

Le panneau d'informations affiche le niveau courant, le nombre de clé, le nombre de vie, le nombre de bombe que le player peut actuellement poser et la portée actuelle du player.

2.5 Gestion des portes

Lorsque le *player* se déplace sur une porte ouvert il est automatiquement transporté vers le niveau auquel la porte doit le mener. Si la porte est fermée et que le *player* à au moins une clé alors la porte s'ouvre est lorsque le joueur appuis sur espace, et le joueur est transporté vers le niveau correspondant.Si le nombre de cle est insuffisant il ne se pas rien.

2.6 Gestion des bombes

Pour gérer les bombes, a crée une structure *struct bomb* composer des variable :
- *int x* : coordonnée x de la bombe,initialiser en fonction la position du player à instant de la pose.
- *int y* : coordonnée y de la bombe,,initialiser en fonction la position du player à instant de la pose.
- *int TIME* : stock le dernier instant de où la bombe à changer état.
- *int range* : portée de la bombe,initialiser en fonction de la range player à instant de la pose.
- *int state* : état de bombe (4 à 1 correspond de la longueur de la mèche (4 pour le plus long, 1 pour le plus cours ; 0 pour l'explosion ; -1 la bombe a exploser).
- *struct player* player* : pointeur vers le player qui a poser la bombe.
- *int exploded* : sert à gérer les explosions,initialiser a 0.
- *int rn,rs,re,rw* : enregistre la portée sur les 4 directions

Et on ajoute au *struct player* les élément *int bombs* : bombs représente le nombre de bombe que le player peut encore poser. Lorsque le player appuis sur *ESPACE*,il pose une bombe.On initialise une *struct bomb*. Tout les secondes, à partir du moment où on poses la bombe, on décrément la *state* si *state*=0 alors on affiche et remplace les cellules de la map en CELL_EXPLOSION en fonction de ça portée et du type de la cellule. Si la cellule est de type CELL_BOXE on on conserve les bits de poids faible lorsqu'on change le type en CELL_EXPLOSION. A la fin de de l'explosion,lorsque *state=-1*, on remplace les CELL_EXPLOSION par CELL_EMPTY si les bits de poids faible sont nul, ou par CELL_BONUS sinon, et on met *exploded* à 1.
Pour gérer les explosions en chaîne, si on explosion touche un bombe (si ça cellule devient de type CELL_EXPLOSION) alors on place ça *state* à 0 se qui fait exploser la bombe.
Pour gérer le plusieurs bombes on a crée un structure *bomb_list* qui est une liste chaînée d'élément *struct bomb*. Cette structure a été ajouter à la *struct map*, ainsi chaque map à ça liste de bombe. Lorsque la liste contient une bombe avec une *state* de -1 alors on la supprime de la *bomb_list* et si d'autres bombes sont en train d'exploser (*state=0* et *exploded=1*) alors on place *exploded* à 2 se qui indique au programme de faire ré-exploser la bombe en fonction des *rn,rs,re,rw*, cela permet de corriger les problème de affichage lorsque deux explosions se croisent.

2.7 Gestion des bonus et malus

on ajoute au *struct player* les élément :
- *int max_bomb* : représente le nombre max de bombe que peut avoir, comprise entre 1 et 9.
- *int range* : représente la portée des bombes, comprise entre 1 et 9.
- *int life* : représente le nombre de vie qu'a le player, comprise entre 1 et 9.

Lorsque de *player* ce déplace sur une CELL_BONUS on fait évolué ces entier en fonction des du bonus.

2.8 Gestion des vies

Lorsque le *player* se trouve dans une CELL_EXPLOSION ou une CELL_MONSTER alors on décrément *player->life*, et on rends le *player* invincible pendant 2.5 secondes.Pour cela on a ajouté à la *struct player* un

élément *int state* qui vaut enregistre l'instant où le *player* de vient invincible, -1 sinon.Lorsque la différence entre instant courant et *state* est supérieur a 2500(ms), on met *state* a -1, le *player* n'est plus invincible.

2.9 Gestion des monstres

Pour gérer les monstres, a crée une structure *struct monster* composer des varialble :
— *int x,y* qui enregistre la position actuelle du monstre.
— *unsigned int lastTime* qui enregistre l'instant du dernier deplacement du monster.
— *enum direction direction* qui stocke la direction actuelle du monstre.
Tout les secondes on choisie aléatoirement une direction, si le déplacement dans cette direction est possible alors on déplace le monstre,sinon seul orientation change. Pour gérer les groupes de monstre, on procèdent de la même manière que les bombes : on ajoute à *Struct map* un élément *struct bomb_list* qui est une liste chaîné d'élément bombes. Si une monstre se trouve dans une CELL_EXPLOSION on le supprime de la bomb_liste, le monstre est mort.

2.10 Fin de partie

Si le nombre de vie passe à zero alors jeu se termine, on affiche une page game over, et le joueur à la possibilité de quitter, recommencer du début ou recommencer depuis la dernière sauvegarde. Si le player se déplace sur la case de la princesse, le jeu se termine et une page le fin de jeu s'affiche.

2.11 Pause

Lorsque que le joueur appuis sur $[P]$ le jeu se met en pause, et un menu apparaît et on stocke dans une variable instant où le player a appuyer sur pause.Si le joueur repend la partie alors on se sert de cette variable pour ajuster $monster->lastTime$ et $bomb->time$ pour que les monstres et les bombes soit dans le même état avant la pause.

2.12 Sauvegarde / Chargement partie

Lorsque le jeu est en pause, le joueur à la possibilité de le faire une unique sauvegarde. La sauvegarde enregistre état du *player* en état de l'ensemble des *map* dans un fichier *saved.txt*. Toutes les bombes posés mais non explosé ne sont pas sauvegardé. Si la bombe est en cours explosions alors la sauvegarde enregistre état de la *map* après l'explosion.
Je joueur a la possibilité de lancer la sauvegarde dans le menu de depart ou dans le menu de game over.