PG110

Oumaima Najib Yahya Mesmoudi

April 2021

1 Introduction

Le but de ce projet est de compléter une ébauche du jeu Bomberman, avec comme langage de programmation le langage C. Dans notre travail, on s'est principalement interessé à la gestion des déplacements du joueur, à l'ajout les monstres dans le jeu, à l'implementation des bombes, et enfin au changement des cartes.

2 Le jeu

Lorsqu'on excécute le jeu, la fonction game_new est appelé dans le main. Cette dernière retourne une structure game ayant les données nénessaires au jeu, à savoir un pointeur vers les cartes et un pointeur vers une structure de type player. Nous avons ajouté par la suite deux pointeurs vers un nouveau type de structure: monster et bomb, qui nous ont permis d'ajouter les fonctionnalités nécessaires à la gestion des bombes et des monstres. C'est dans cette fonction que sont initialisées et sauvegardés toutes les données du jeu : les 8 maps, les monstres relatifs à chaque map, et les bombes. Une autre fonction était nécessaire pour actualiser l'etat du jeu : game–update, à ce niveau sont mis à jour les différents paramètres relatif au jeu.

3 Déplacement

Au début, le joueur peut se déplacer à sa guise : il peut se déplacer meme à travers le décor et sortir de la map. Le but étant que le joueur respecte le décor de la map, reste dans le cadre de cette dernière et puisse déplacer des boites. Les fonctions responsables du déplacement du joueur sont player_move et player_move_aux. Dans la fonction player_move_aux, il y a une boucle switch qui prend en argument le type de la case, à chaque type est assigné une certaine valeur, c'est cet entier que l'on est censé utiliser dans la fonction player_move.Cependant, on a préféré utiliser directement la fonction player_move, en effet, on a ajouté des conditions pour chaque cas de déplacement, pour pouvoir changer les coordonnées du joueur, il faut que celles-ci soient incluses dans la grille de la map,

de plus, il faut que la case du prochain déplacement du joueur ne soit pas de type CELL–SCENERY. Le joueur peut aussi déplacer une box à la fois si rien ne le gène dans la poussée.

4 Les bombes

On a créé le fichier bomb.c pour la gestion des bombes. Ce dernier contient la structure bomb, il s'agit d'une liste chaînée, indispensable au dépot simultanée de plusieurs bombes. La fonction bomb_init, qui renvoie une structure bombe, est appelé lors l'initialisation du jeu, dans le game_new. La structure permettant le dépôt des bombes étant implémentée, comment faire pour déposer une bombe lorsqu'on appuie sur la touche espace ? Grace aux fonction bomb_add et bomb_meca ; bomb_add est une fonction qui ajoute un élément au début de la liste chaînée.

Cette fonction est appelée en appuiant sur la touche espace: elle ajoute une bombe en tête de liste, cette bombe a pour drop_time l'instant ou l'on a appuyé sur espace, et les positions respectives de la bombes sont alors les positions du joueur à ce moment là. Quant à la fonction bomb_meca, elle est appelé tout le long du jeu. Cette dernière prend en argument la liste chainée des bombes reliés au game, et agit de telle façon que tant que la liste ne contient pas 1 et un seul élément, qui est l'élément ajouté au tout début du jeu grace à bomb_init, on initialise un timer pour chaque bombe et on déroule l'explosion.

4.1 Gestion de l'explosion

4.1.1 Explosion des boites

On a assigné à chaque boite un sous-type qui donne une information sur le bonus ou mallus que contient la boite.

4.1.2 Bonus et Malus

À la fin du timer, il devait etre possible de remplacer une case explosion en la case bonus définissant le contenue de la boite qui s'est faite exploser. Pour se faire, il était nécessaire de garder une information sur le sous-type de la boite se trouvant dans la portée de la bombe: on a alors ajouté des sous-types au type CELL_EXPLOSION. Grace à cette petite astuce, il est possible de garder une information sur le bonus à afficher malgré que l'on ait changé le type de la cellule.

4.1.3 Perte de vie du joueur

Afin de diminuer la vie du joueur lorsque sa position correspond à une cellule explosion, on a pensé à jouer sur le timer de la fonction bomb_meca; on a ajouté à l'étape finale du timer une condition sur ce dernier, sur un intervalle de 40 secondes, la fonction player_gets _harmed est appelé , grace à elle, le joueur perd une vie.

5 Monstres

De la même manière que pour les bombes, on a créé le fichier "monster.c" pour la gestion des monstres.

Le choix d'une implémentation basée sur la structure d'une liste chaînée était motivé pour pouvoir décider librement le nombre des monstres tout en utilisant le moins d'espace mémoire possible, on a tenté d'optimiser encore plus cette implémentation en supprimant à chaque fois qu'un monstre meurt sa case correspondante dans la liste mais faute de temps disponible et de notre avancée générale dans le projet on n'a pas pu aboutir à cette solution.

La structure monster contient en outre la position du monstre en question, le niveau correspondant à ce dernier dans le champs "monster.level" et un paramètre indicateur de vie dans le champ "monster.monster_is_alive" . Ces deux derniers paramètres sont systématiquement vérifiés pour l'affichage des monstres dans la carte via la fonction "monsters_display". Au chargement de chaque carte et à l'aide de la fonction "create_monsters_fom_map", la carte est parcouru cellule par cellule et à chaque passage par une "CELL_MONSTER" on initialise un nouveau monstre dans la case en ajoutant un nouveau élément à la fin de la liste chainée, en ce qui concerne les déplacements, on détermine dans la fonction "game_update" quand est-ce qu'on doit faire appel à la fonction "monsters_next_move" qui prépare les monstres de la carte en question à faire le prochain mouvement aléatoire.

5.1 Disparition des monstres

6 Maps

Puisque le nombre des cartes est fixé à 8 par le codage on a créé un tableau de 8 élément pour l'ensemble des cartes dans la structure game, au lancement du jeu chaque carte est chargée à partir du fichier .txt correspondant à l'aide de la fonction " read_map" et enregistrée par la suite dans le champ "game.maps[N° de la carte]", ce qui permets d'enregistrer toute modification faite sur une carte dans une partie pendant toute la durée de ce dernière, lorsque le joueur entre d'une porte, la carte qui est se trouve au premier plan va changer en fonction de l'encodage de la porte.

7 Portes

Chaque porte dans la carte donne accès soit au niveau qui précède ou celui qui succède le niveau actuel sur le jeu. L'encodage de chaque porte détermine où le joueur va aller : une fonction spécifique est incrémentée à chaque passage par la porte, ce qui permets d'appeler soit "game_next _map" soit " game_previous_map"

8 Gestion des vies