**PROJET BOMB’EIRB**

(Michel Vann & Benjamin Scialom)

1. **Gestion des déplacements**

* La gestion des déplacements se fait dans les le fichier **player.c** et plus précisément grâce aux fonctions suivantes :
* box\_move et box\_move\_aux
* player\_move et player\_move\_aux
* player\_from\_map

1. **Chargement des cartes**

* Le chargement des cartes s’effectue grâce à la fonction fill\_map appelée dans la boucle “for” de game\_new. Cette fonction lit chacun des fichiers situés dans le dossier data. Cela consiste essentiellement à parcourir le fichier et à associer aux numéros écrits le type de la cellule. Les différents types des cellules sont implémentés dans un enum compose\_type. Pour différencier les types, on joue donc sur les bits de poids fort et les bits de poids faible.

1. **Gestion des portes**

* La gestion des portes s’effectue dans la fonction player\_move\_aux. Dans le cas où le joueur rencontre une porte fermée, il peut ouvrir la porte si le joueur à au moins une clef dans son inventaire. Si la porte est ouverte alors il passe vers le niveau indiqué par la porte. Les numéros des portes sont précisés dans le enum compose\_type du fichier **map.h**.

1. **Gestion du panneau d’information**

* La gestion du panneau d’information se fait grâce à la fonction game\_banner\_display qui se situe dans le fichier source **game.c.** On y définit les sprites que la fonction doit afficher en précisant la position. La bannière s’adapte automatiquement à la taille de la carte.

1. **Gestion des bombes**

* Pour la gestion des bombes, nous avons utilisés une liste chaînée. L’ajout de nouvelles bombes se fait toujours en fin de liste. Le fichier **bomb.c** permet la gestion d’une seule bombe alors que le fichier **list\_bomb.c** permet de faire la liaison avec la liste chaînée. Afin de gérer la diminution de la mèche, il y a un cas CLOCK dans la fonction bomb\_display qui diminue la mèche grâce à SDL\_GetTicks(). Le cas BOOM génère l’explosion grâce à la fonction bomb\_explosion\_display.

1. **Gestion bonus et malus**

* Nous avons mis les différents types de bonus dans un enum bonus\_type. Au traitement d’une case cell\_bonus, la fonction display\_bonus dans **map.c** permet de différencier les différents types de bonus.

1. **Gestion des vies**

* Le cas du game over se fait dans la fonction game\_display. S’il n’y a plus de vie alors la variable game->jeu passe à l’état GAME\_OVER. Cela a pour incidence de faire afficher un sprite de défaite. Cette option est implémentée dans game\_update.

Pour l’incrémentation et la décrémentation des vies, cela se fait grâce à des fonctions implémentées dans player.c qui sont :

* player\_inc\_nb\_life
* player\_dec\_nb\_life

De plus le joueur clignote lorsqu’il atteint par une bombe qui explose ou toucher par un monstre. Pendant le temps où le joueur clignote, ce dernier est invincible.

1. **Gestion des monstres**

* Nous avons là aussi utilisé une liste chaînée. Les monstres sont chargés dès le départ du jeu dans game\_new grâce à la fonction game\_monster\_from\_maps qui est dans **game.c**. La vitesse des monstres est gérée grâce au numéro de la map sur laquelle le joueur est situé. En effet, on diminue le délai de déplacement proportionnellement au numéro de la map. Cela est implémenté dans la fonction monster\_init de **monser.c**. Les monstres protègent la princesse s’ils sont à une distance de deux cases maximum dans leur champ de vision.

1. **Fin de partie**

* La fin de partie est enclenchée lorsqu’on est sur la princesse. L’état de la variable game->jeu a alors pour valeur GAME\_WIN. Une image est alors affichée pour signifier que la partie est gagnée. Cela est géré par game\_update qui est dans **game.c**. Pour retourner dans le menu du jeu, il faut appuyer sur la touche tabulation.

1. **Pause**

* Une variable pause a été ajoutée dans la structure game. Si elle vaut 1 la pause est enclenchée sinon le jeu continue. A chaque appuie de la touche p, la variable vaut 1 si elle est à 0, et inversement sinon. Si pause = 1, une image en haut à gauche de la fenêtre est affichée.

1. **Sauvegarde**

* La sauvegarde se fait à partir de deux fonctions qui sont :

- save\_game (touche s)

- load\_save (touche l)

La première fonction permet d’écrire dans un fichier .txt les numéros

correspondants à chaque cellule. Il faut aussi récupérer les paramètres du joueur qui sont écrit après la map dans ce même fichier.

La seconde fonction ressemble beaucoup à fill\_map sauf que l’on retourne rien. On fait en sorte que la carte du niveau dans lequel on était affiche la carte sauvegardée (save.txt) grâce à la fonction save\_game. Il y a aussi mise à jour des paramètres du joueur (toutes les fonctions player\_set …).